Support and Rationale Document

for the

Software Communications Architecture Specification

APPENDIX D: NETWORKING SUPPORT AND RATIONALE

June 30, 2000

Table of Contents

1 INTRODUCTION

The SCAS does not contain a major section on the requirements and design for the networking architecture, hence the body of the SRD contains no specific section on rationales for networking decisions. Instead, networking is embodied in the requirements contained in the software section of the SCAS (section 3), which are accordingly addressed in section 3 of the SRD.

The material contained in section 2 of this appendix describes how the Networking Working Group evaluated various options for a networking architecture and arrived at the overall implementation of the Networking API Transfer Mechanisms and Networking API Service Definitions.

Section 3 of this appendix contains additional details of the analyses of the networking options.

2 NETWORKING OVERVIEW

The MSRC divided the Step 2A work among multiple Integrated Product Teams (IPTs). The Architecture IPT was defined to work the specific task issues related to the architecture. Working groups were chartered by the Architecture IPT to complete the analysis and definition of the JTRS architecture. Key among these was the Core Framework (CF), Security, Use Case, and Networking Working Groups (NWG). The charter of the NWG was the development of standard interfaces within the JTRS that would allow external protocol resources to communicate as distributed resources. The JTRS paradigm for this distributed component interoperability required that the communication between the layers of the generic OSI network stack (see Figure 1) had to be revisited.

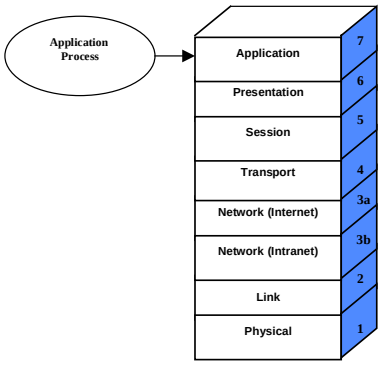


Figure 1. OSI Stack

This meant that the external network protocol stack would have to be capable of being distributed within the Joint Tactical Radio as shown in Figure 2. The capability to effectively distribute the stack depends on the ability to define an interface (service definition) and appropriate transfer mechanism between the distributed network stack entities. The JTRS Operational Requirements Document (ORD) factors of interoperability, performance, security, and portability were considered the most important requirements in determining the process, rationale and final outcome of a definition for effectively distributing these interfaces within a JTRS.

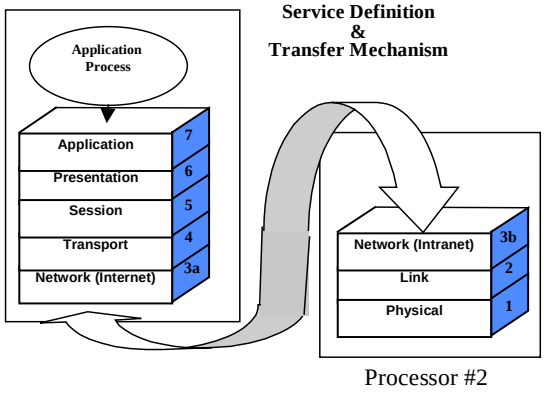


Figure 2. Distributed OSI Stack

The members began work in December 1999 to set tasks for the working group. This was followed by the start of teleconference meetings in January of 2000. Initial discussions started directly from the Architecture Definition Report and led to the development of a Networking Application Programming Interface (API) that attempted to abstract the functionality required at the Modem, Link, and Network layer into a small group of operations required for status, control, and data flow. This Networking API was considered the first definition of an Networking API Service Definition (NSD) and followed from the work of the GloMo APIs. Work then began on developing alternates for the Networking API Transfer Mechanism (NTM). As part of this effort it became apparent that the NTM must cover four separate areas. These were: 1) resource set up and initialization, 2) non real-time resource control, 3) real-time resource control, and 4) data flow.

After further discussions both within the Networking Working Group and externally with the Joint Program Office and other members of the MSRC during Internal Program Review (IPR) #3, the team instituted a rigorous process for analysis of both the NSD and NTM alternatives. The process involved the framing of the problem to be solved, derivation of the Figures of Merit (FOM), criteria for evaluating selected options for the NSDs and NTMs, selection and analysis of options, and then review, rating, and selection of the best approaches.

2.1 NETWORKING WORKING GROUP CHARTER

The charter of the NWG was the development of standard interfaces within the JTRS that would allow external protocol resources to communicate as distributed resources.

2.2 NETWORKING WORKING GROUP PROBLEM STATEMENT

The problem statement adopted by the NWG is shown below:

Determine an effective interface definition for distributing the JTRS network communication protocol stack components within a Joint Tactical Radio System that takes into account interoperability, performance, security, extensibility, and portability. This definition should take into account the Networking API Service Definition, the Networking API Transfer Mechanism, where the Networking API Service Definitions and Transfer Mechanisms are documented, and how configuration control for the Networking API Service Definitions and Transfer Mechanisms will be managed. This includes how the Networking API Transfer Mechanism(s) and the Networking API Service Definition(s) support the ORD's concept of component interoperability and future plug and play capability.

2.3 NETWORKING WORKING GROUP ANALYSIS PROCESS

The following diagram details the Networking Working Group analysis and task process for the development of networking components for the JTRS SCA.

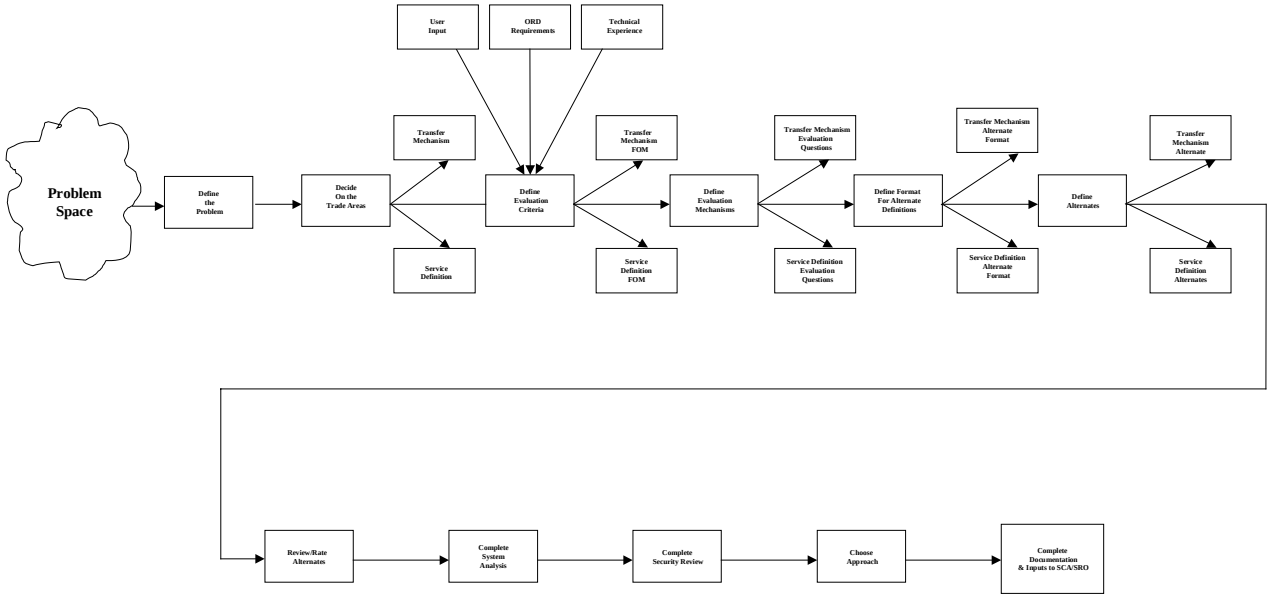


Figure 3. Networking Working Group Analysis Process

2.4 NETWORKING WORKING GROUP DECISION PROCESS

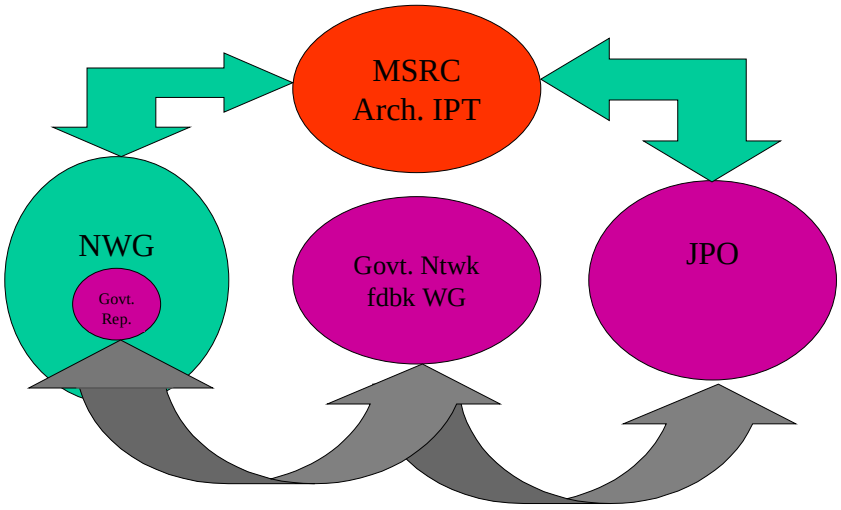


Figure 4. Networking Working Group Decision Process

2.5 NETWORKING WORKING GROUP TOP-LEVEL DEFINITIONS

In the course of the work of developing a suitable definition, a set of terms was defined to clarify the meaning of references for the NWG and for discussions with others. These terms are provided below to assist with the understanding these NWG analyses.

Networking API: The interface definition between a JTRS waveform protocol layer service provider and a service user. It covers the JTRS waveform Modem, Link, and Network interfaces. For example, a Modem API is an interface between a modem and the application using it, whether that application is a layer of an OSI networking protocol, a voice application, or some other application.

Networking API Instance: A Networking API Instance is a coupling of a Networking API Service Definition and a Networking API Transfer Mechanism for an implementation. The Networking API Instance provides the mechanism for distributing the waveform layers within a single processor or across different processors. (Note that Networking API and Networking API Instance are often used interchangeably.)

Networking API Service Definition (NSD): The service definition for a waveform protocol layer details the operations (primitives), the attributes (variables), their representation (structures, types, formats) and its behavior.

Networking API Transfer Mechanism (NTM): The transfer mechanism provides the communication between the JTRS waveform protocol layer service provider and a service user. CORBA and STREAMS are examples of transfer mechanisms.

Waveform: Waveform defines the communications between two JTRS radios and includes modulation as well as OSI protocols layers 2-4.

The relationship of the Networking APIs, NSD, and NTM to the External Network Protocols and the SCA JTR is shown in Figure 3. The system communicates with peer systems over wireless and wireline protocols. Note that JTRS is called the SCA Radio System in the figure.

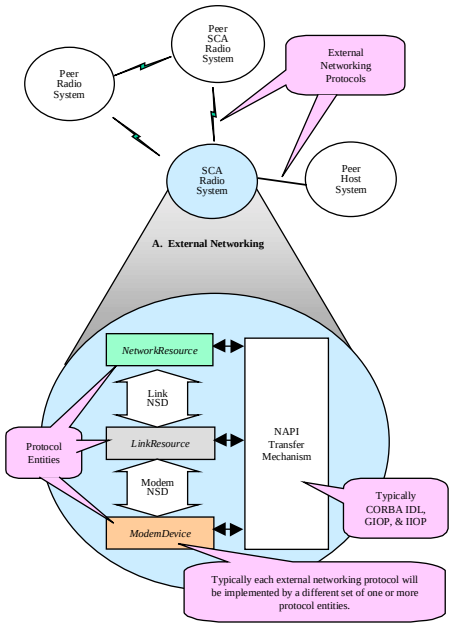


Figure 5. External Network Protocols and SCA Support

2.6 NETWORKING WORKING GROUP ANALYSIS

2.6.1 Criteria Derivation

Table 1 lists sections of the Joint Tactical Radio System (JTRS) Operational Requirements Document (ORD) that are relevant to Networking APIs. The Networking API requirements were then derived from these relevant sections. (Both the currently approved ORD, 23 Mar 98, and a recent draft ORD, 9 Feb 00, were examined with no differences in the derived Networking API requirements.) Each derived Networking API requirement is categorized according to the

category of the JTRS ORD section it was derived from and whether it is a primary or secondary intent of the JTRS ORD section: Key Performance Parameter (KPP) primary, Threshold primary, Objective, KPP secondary, Threshold secondary, or Recommendations (other than KPP, Threshold, or Objective).

If a Networking API requirement can be derived from multiple ORD sections, then it is categorized according to the highest categorized source section according to the order listed above. Table 2 lists the derived Networking API requirements, their category, and a cross reference to the source ORD sections and their categories.

JTRS ORD paragraph 4.a.(1)(b) calls out an "an internal growth capability through an open systems architecture approach in compliance with the Joint Technical Architecture" (JTA) that "shall be modular, scaleable, and flexible." The JTA calls out the Defense Information Infrastructure Common Operating Environment (DII COE) as a requirement. The DII COE has requirements for Application Programming Interface (API) standards for applications (namely Unix Sockets and Windows WINSock). However, the DII COE does not have an equivalent set of API standards for wireless and networking protocols at lower layers. A modular, wireless, networking, open systems architecture requires such APIs. Thus, the need for Networking APIs at these lower layers was identified during JTRS Step 1.

A Networking API instance consists of two elements: a Networking API Service Definition and a Networking API Transfer Mechanism. There are multiple alternatives for each of these two elements. The derived Networking API requirements can be used as figure of merit (FOM) criteria to analyze and evaluate the different Networking API alternatives.

Table 3 allocates the derived Networking API requirements to the Networking API Service Definition and Networking API Transfer Mechanisms. The derived Networking API requirement category is listed. Evaluation weights were developed using the derived Networking API requirement category with adjustments based upon our Networking Integrated Product Team (IPT) experience.

Table 1. JTRS ORD and Derived Networking API Requirements.

JTRS ORD Sections Relevant to Networking APIs	Derived Networking API Requirements	Category
4.a.(1)(a) The JTRS architecture shall be capable of supporting secure and non-secure voice, video and data communications using multiple narrow-band and wide-band waveforms for KPP, threshold, and objective at Tables 1 and 2, including future waveforms as they are developed.	• Support legacy networking and non-networking waveforms	K
	• Support future networking and non-networking waveforms	T
4.a.(1)(b) The JTRS program shall provide an internal growth capability through an open systems architecture approach in compliance with the Joint Technical Architecture, and shall be modular, scaleable, and flexible.	• Support portability • Provide networking APIs equivalent to application APIs specified by DII COE • Support component interoperability	K
	• Support extensibility	K secondary
4.a.(1)(c) Each JTR set shall provide the operator with the ability to load and/or reconfigure its modes/capabilities (via software) while in operational environment.	• Support Networking API transfer mechanism dynamic construction and deconstruction	K
4.a.(2)(i) Shall provide for INFOSEC and protection of data in Secret High network	• Support Security	T

JTRS ORD Sections Relevant to Networking APIs	Derived Networking API Requirements	Category
4.a.(2)(j) Each JTR set shall interface with and support functions of cryptographic systems listed in Annex D.	• Support security	T
4.a.(3) The JTRS shall be capable of providing scaleable networking services for connected RF (over the air) networks, host networks, and hybrid networks	• Support networking waveforms	T
	• Need to adapt to commercial trends (since many host networks will evolve as commercial trends evolve)	T secondary
4.a.(3)(i) The JTRS network shall have the capacity to meet the information flow of waveforms/ capabilities as specified at Tables 1 and 2 and meet the information flow required by new capabilities and latency near zero.	• Need performance (efficiency) to support waveforms, including high speed legacy waveforms	T
	• Need performance (efficiency & latency) to support waveforms including high speed new waveforms	O
5.c.(1) JTR must be easily maintainable and operable, incorporating the principles of modularity and commonality.	• Need to have low life cycle costs • Support portability • Support extensibility • Support reuse at same layers • Support component interoperability	R
5.h.(1) The JTR acquisition will adhere to the JTA in identifying the standards and guidelines.	• JTA calls out DII COE has API requirements for application protocols	R
5.h.(2) Only NSA endorsed and approved security products, techniques, and protective services shall be used to secure classified communications.	• Support Security	R
7.a The system shall be developed incrementally providing increased capabilities as it matures.	• Support SCA incremental releases • Support Extensibility	R
1.a. 5th sentence The JTR must operate with legacy equipment ... and incorporate new waveforms as they are developed.	• Support legacy and future waveforms • Need to adapt to commercial trends • Support extensibility	R
1.a. 6th sentence The family of radios will be scaleable by virtue of form, fit and cost to meet specific user operational needs.	• Need to be low-cost • Support portability • Support component interoperability	R
1.a. 7th sentence The JTR will also provide growth capability through an open system architecture that enables technology insertion through evolutionary acquisition or P3I.	• Support future waveforms • Need to adapt to commercial trends	R
1.a. 8th sentence The JTR will be capable of higher channel data throughput rates.	• Support performance needed for high throughput waveforms	R
1.d. 1st sentence The JTR will be supported by commercial sources and practices to provide the most cost effective solution.	• Need to be low-cost	R
	• Need to obtain commercial acceptance	R

Note: K = KPP; T = Threshold but not K; O = Objective, R = Other ORD Recommendations

The following table depicts the Networking API requirements, associated categories, and source sections from the JTRS ORD.

Table 2. Networking API Requirements, Categories, and Source ORD Sections

Networking API Derived Requirement	Networking API Category	**Source ORD Section**	**Source ORD Category**
Need for Networking APIs	K	4.a.(1)(b)	K
		5.h.(1)	R
Support Portability	K	4.a.(1)(b)	K
		5.c.(1)	R
		1.a. 5th sentence	R
Support New Waveforms	T	4.a.(1)(a)	T
		4.a.(3)	T
		1.a. 7th sentence	R
Support Legacy Waveforms	K	4.a.(1)(a)	K
		4.a.(3)	T
Support Performance	T	4.a.(3) (efficiency)	T
		4.a.(3) (latency)	O
		1.a. 8th sentence	R
Support Extensibility	K (secondary)	4.a.(1)(b)	K (secondary)
		5.c.(1)	R
		7.a.	R
Support Reuse at Same Layer	R	5.c.(1)	R
Support Dynamic Construction/ Deconstruction	K	4.a.(1)(c)	K
Support Component Interoperability	K	4.a.(1)(b)	K
		5.c.(1)	R
		1.a. 5th sentence	R
Support Low-Cost	R	5.c.(1)	R
		1.a. 5th sentence	R
		1.d. 1st sentence	R
Foster Commercial Acceptance	R	1.d. 1st sentence	R
Support Security	T	4.a.(2)(I)	T
		4.a.(2)(j)	T
		5.h.(2)	R
Adapt To Commercial Trends	T (secondary)	4.a.(3)	T (secondary)
		1.a. 7th sentence	R
Support SCA	R	7.a.	R

Table 3. Deriving the Weighting of the Evaluation Criteria

Allocation to Networking API Service Definition	Allocation to Networking API Transfer Mechanism	Highest Category	Weighting Value (Note 1)
Support Portability	Support Portability	K	20% (Note 2)
Support New Waveforms	not applicable	T	10%
Support Legacy Waveforms	not applicable	K	15%
not applicable	Support new & legacy waveforms	K	15% (Note 3)
Support Performance	Support Performance	T	10% (service) 15% (transfer) (Note 4)
Support Extensibility	Support Extensibility	K (second-ary)	5%
Support Reuse at Same Layer	not applicable	R	5%
not applicable	Support Dynamic Construction/ Deconstruction	K	10% (Note 5)
Support Component Interoperability	Support Component Interoperability	K	15%
Support Low-Cost	Support Low-Cost	R	10% (Note 6)
Foster Commercial Acceptance	Foster Commercial Acceptance	R	5%
Support Security	Support Security	T	Binary (Note 7)
Adapt To Commercial Trends	Adapt To Commercial Trends	R	5%
Support SCA	Support SCA	R	Binary (Note 8)

Note 1: Preliminary weight assignments: K (primary rqmt) = 15%; T (primary rqmt) = 10%; K & T secondary requirements = 5%; and O & R = 5%.

Note 2: Increase weight by 5% due to importance of portability as rationale behind creating JTRS to reduce efforts of creating & using waveforms.

Note 3: Combined legacy and new waveform requirements for transfer mechanism analysis since this area is impacted much more by service definition choices

Note 4: Increased weight transfer due to importance of transfer on performance.

Note 5: Decreased weight transfer to support increases discussed in notes 2 and 4.

Note 6: Increase weight due to importance of cost to JTRS as a major program driver.

Note 7: Chose to make this a binary yes/no since NSA acceptance is a go/no-go.

Note 8: Chose to make this a binary yes/no since Networking APIs have to work within SCA framework.

2.6.2 Networking API Service Definition Analysis

2.6.2.1 Networking API Service Definition Criteria (Figures of Merit) and Ratings

(Note: the criteria numbers in the following tables refer to the detailed questions presented in section 3 of this document. For example, criteria 6.1 (Waveform Portability) is treated in sections 3.1.n.6.1, where n refers to Service Definition Option n (1-4).)

Table 4. Networking API Service Definition Figure Of Merit (FOM) Analysis

Criteria	Weight	Option 1	Option 2	Option 3	Option 4	Option 5
6.1 Waveform Portability	20%	9	3	9	9	9
6.2 Ability to support future waveforms	10%	9	3	9	9	9
6.3 Ability to support legacy waveforms	15%	9	9	9	9	9
6.4 Performance (Efficiency/Latency)	10%	3	3	9	9	9
6.5 Extensible	5%	9	1	9	9	9
6.6 Networking API Re-use at Same Layer	5%	3	9	1	3	3
6.7 Component Interoperability (Component plug & play)	15%	9	9	9	9	9
6.8 Cost	10%	3	1	9	9	3
6.9 Commercial Acceptance	5%	3	1	3	3	3
6.10 Security	**binary (1 or 0)**	3	9	1	3	3
6.11 Ability to adapt to commercial trends	5%	9	1	9	9	9
6.1.3 Impact to SCA	**binary (1 or 0)**	9	9	9	9	9
SCORE	100%	**7.2**	**4.6**	**8.3**	**8.4**	**7.8**

Waveform Portability	Ability of waveform software/hardware to move from one JTRS platform to another without modification (other than a potential recompile).
Future support	Ability of the system to support future waveforms in a consistent manner (i.e. that same manner as new waveforms are supported). This includes the ability to support networking for multimedia services.
Legacy support	Ability of the system to support existing waveforms (both networking and non-networking) in a consistent manner (i.e. that same manner as new waveforms are supported).
Performance	Ability of the system to support high speed, high data, low latency waveforms. Includes high data rate throughput and low command latency from red processor to modem cards.
Extensible	Ability of the architecture to be extended to support new waveform interfaces and mechanisms.
Re-use	Ability of the Networking API definition to be re-sued at the same layer, e.g. use the same modem Networking API definition for multiple (different) modems.
Component interoperability	Ability of individual waveform components to be reused by other waveforms; the ability of components to support plug and play.
Cost	Protocol software entity development, maintenance, and porting cost (Higher score corresponds to lower cost.)

Commercial Acceptance	Degree to which the architecture is acceptable to the commercial world.
Security	Ability of the design to support the JTRS security architecture. This point is considered binary in that the alternate will either support or not support the JTRS security concept. *Difficulty in implementing with the alternate is quantified under Cost.
Commercial Adaptability	The ability of the architecture to adapt to emerging commercial networking standards.
Support of Core Framework	Compliance of the design to the Core Frame Architecture.

Ratings to Detailed Questions

The Combined Answers to the Detailed Questions are then Combined to yield a single 1, 3, or 9 value in the Combined Evaluation Factor

	Option 1	Option 2	Option 3	Option 4	Option 5
6.1.1	9	9	9	9	9
6.1.2	3	1	9	9	3
6.2.1	9	3	9	9	9
6.2.2	9	3	3	9	9
6.2.3	9	9	9	9	9
6.2.4	9	1	9	9	9
6.3.1	3	9	9	9	3
6.3.2	9	1	9	9	9
6.3.3	9	9	3	9	9
6.3.4	9	3	9	9	9
6.3.5	9	9	9	9	9
6.4.1	3	3	9	9	9
6.4.2	3	3	9	9	9
6.4.3	3	9	9	9	3
6.5.1	9	1	9	9	9
6.5.2	3	9	3	9	9
6.6.1	3	9	1	3	3
6.6.2	3	9	3	9	9
6.7.1	3	9	3	9	9
6.7.2	9	9	9	9	9
6.8.1	3	1	9	9	3
6.9.1	3	1	3	3	3
6.9.2	1	1	3	9	3
6.10.1		9			
6.11.1	9	1	9	9	9
6.1.3.1	9	9	9	9	9
6.1.3.2	9	9	9	9	9

2.6.2.2 Networking API Service Definition Options

The following sections provide summary information for the analysis that the NWG did for the Networking API Service Definition. The group derived five options for the service definition that cover the range of possible implementations and then analyzed each using questions developed from the evaluation described in Section 2.6.2.1. The ratings of answers provided the basis of the selection of the chosen option for the SCA.

2.6.2.2.1 Service Definition Option #1 Name/Value Pair

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.1.1.

2.6.2.2.1.1 Description

This option provides a small service definition since it defines a fixed set of operations that all Networking API Service Definitions could draw from. In particular, the Modem Networking APIs could draw from a set of eight possible operations while the Link, Network, and Transport Networking APIs could draw from a set of twelve possible operations. A small number of operations helps to develop consistency between Networking API Service Definitions and facilitates going from one waveform's service definition to another. Some of the operations can be reused from the Core Framework Lifecycle Interface thus reducing the development effort and possibly the implementation footprint required. This option defines a method of interface configuration control, interface version identification, and interface negotiation through the use of unique identifiers, such as CORBA's UUID. It was designed to give vendors and users great freedom and flexibility in trading off performance, portability and cost drivers. It promotes open interface standards for waveforms and allows customers to assess the impact of changes. Its configuration control provides a mechanism for managing evolution and tracking changes. The flexible service definition as well as the configuration control approach was modeled after the same process that is used to specify and manage MIBs for SNMP and CMIP.

2.6.2.2.1.2 Architectural Considerations

This option requires the Architecture and Core Framework to provide a common mechanism to identify the Networking API Service Definition Name/Value Pairs to insure networking resources have a common interface.

2.6.2.2.1.3 Relative Strengths and Weaknesses

This option has the following strengths:

1. Relatively low implementation costs.
2. Easily extensible interfaces.
3. Flexible interfaces.
4. Most compatible option to the CF (can reuse the LifeCycle interface).

This option has the following weaknesses:

1. High overhead (with impact on throughput) for the CORBA "Any" implementation of name/value pairs as contrasted to other CORBA IDL mechanisms.
2. Higher maintenance costs with second level name/value pair definitions (as compared to maintaining interfaces defined purely in IDL).
3. May not map to existing commercial API standards.

2.6.2.2.2 Service Definition Option #2 Super Networking API Service Definition

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.1.2.

2.6.2.2.2.1 Description

The Super Networking API Service Definition is a term used to describe the goal of defining a single universal interface for all waveform applications. Each level; Network, Link, and Modem would have its own Super Networking API Service Definition and all JTRS compliant applications would use these Networking API Service Definitions when crossing these OSI boundaries. Using the Super Networking API Service Definition approach, a single interface for each layer of the network stack would be established. This interface would have to be robust enough to support all existing waveforms and all future waveforms targeted for the JTRS product line. The Super Networking API Service Definition is hard, fast and well defined, meaning all interfaces would have a defined data type. The only way to change a Super Networking API service Definition would be to update the interface with each revision of the SCA. New waveforms would be forced to operate within the boundaries of the existing Networking API Service Definitions.

With respect to the Link and Network the Super Networking API Service Definition approach may meet the needs of the JTRS user community. The utility of the Super Networking API Service Definition approach with respect to the modem interface is questionable.

The Modem Super Networking API Service Definition presents several problems:

1. There are a variety of modem products available. The configuration and control interfaces required by the Link layer software to ensure an orderly flow of data are often low-latency interfaces and unique to each hardware device.
2. There are a variety of waveforms that need support. The Modem Super Networking API Service Definition will most likely be required to support all JTRS waveforms, both networking and non-networking. The Modem Super Networking API Service Definition would therefore need to be inclusive of all waveform unique interfaces.
3. Each of the interfaces defined in IDL will impact the Core Framework (CF)

2.6.2.2.2.2 Architectural Considerations

This option requires the Architecture and Core Framework to provide a common mechanism to identify Networking API Service Definitions to insure networking resources have a common interface.

2.6.2.2.2.3 Relative Strengths and Weaknesses

This option has the following strengths:

1. The use of a Super Networking API Service Definition would foster reuse of service definitions amongst multiple waveforms.
2. A single API Service Definition for each layer for the protocol stack would foster greater “mix and match” portability between various vendors and waveforms.

This option has the following weaknesses:

1. A single Super Networking API Service Definition for each OSI layer would be inclusive of interfaces required for all ORD supported waveforms for each OSI layer, this would make the API very complex.
2. It is unlikely that a single Super Networking API Service Definition for each OSI layer would gain wide commercial acceptance.
3. The Super Networking API Service Definition approach would require a network component to support all the interfaces be defined by the Super Networking API Service Definition. This would add complexity to the development of network components.
4. Since a Super Networking API Service Definition would have to encompass the interfaces required for all ORD supported waveforms, as new waveforms were developed the existing Networking API Service Definition would have to be updated to include any new interface requirements. This would make the Super Networking API Service Definition difficult to maintain.

2.6.2.2.3 Service Definition Option #3 Commercial Model

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.1.3.

2.6.2.2.3.1 Description

This alternate supports the commercial world approach to creating/controlling Networking API Service Definitions (similar to that used for controlling MIBs in the SNMP protocol). This alternate encourages total reuse of Networking API Service Definitions. The Networking API Service Definition must be reused in its entirety. If an API Service Definition is not available to reuse in its entirety a new Networking API Service Definition must be created (may be a partial copy of the previous API Service Definition). For example if an API Service Definition is available for SINCGARS and if this API Service Definition can be reused for all SINCGARS implementations, then all SINCGARS implementations are identical from a portability perspective. However if an SINCGARS implementation has a feature that is not supported by the current documented SINCGARS API Service Definition, a new SINCGARS API Service Definition must be written. The "Commercial Model" is accomplished by the following methods:

1. Specifying methods to control the growth of API Service Definitions via configuration control body.
2. Providing a method for API negotiation/identification (i.e. Universally Unique Identifier [UUID].

2.6.2.2.3.2 Architectural Considerations

This option requires the Architecture and Core Framework to provide a common mechanism to identify Networking API Service Definitions to insure networking resources have a common interface.

2.6.2.2.3.3 Relative Strengths and Weaknesses

This method has the following strengths:

1. Provides for reuse and consistent interfaces for waveform implementations which can support a common interface.
2. Provides for future waveforms through copying and modifying of the existing Networking API Service Definition.

The method has the following weaknesses:

1. It requires a control body to maintain recommended common interfaces and to decrease the propensity for growth.
2. It requires reuse of the Networking API Service Definition in its entirety; lacks inheritance.

2.6.2.2.4 Service Definition Option #4 Commercial Model with Inheritance

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.1.4.

2.6.2.2.4.1 Description

This alternate supports the flexibility of how the commercial world creates Networking API Service Definitions combined with a configuration control process (similar to that used for controlling MIBs in the SNMP protocol) to support interoperability and portability. This option, which is an object-oriented approach, facilitates polymorphism and code reuse via common Networking API Service Definitions. The granularity of a Networking API Service Definition determines its reusability. For example a Networking API Service Definition for a F/A - 18 aircraft would be difficult to reuse for anything except for fixed wing fighter aircraft. However if you break the Networking API Service Definition into smaller inheritable pieces, those pieces (e.g. throttle, sensors, displays, navigation, communication,...) can be reused by many aircraft (fixed and non-fixed wing). The same concept applies to Networking API Service Definitions, where there are functions that are common to a protocol layer (e.g. flow control, packet data exchange, acknowledgment processing...). The small pieces are called the building block APIs. To promote commonality of the building blocks Networking API Service Definitions, a control and identification method is required. "Commercial Model with Inheritance" is accomplished by the following methods:

1. Mandating that Networking API Service Definitions be documented in a standard language which supports inheritance (i.e. Interface Definition Language [IDL])
2. Specifying methods to control the growth of inheritance trees
3. Providing a method for API negotiation/identification (i.e. Universally Unique Identifier [UUID]
4. Providing the framework for creating building block API Service Definitions that can be inherited by API Service Definitions to promote common interfaces.

2.6.2.2.4.2 Architectural Considerations

This option requires the Architecture and Core Framework to provide a common mechanism to identify Networking API Service Definitions to insure networking resources have a common interface.

2.6.2.2.4.3 Relative Strengths and Weaknesses

This method has the following strengths:

1. Provides consistent interface definition method.
2. Provides for extensibility.
3. Provides for future waveforms to inherit/reuse existing waveform Networking API Service Definitions.
4. Fosters backward compatibility.

The method has the following weaknesses: it needs a control body to maintain recommended inheritance trees to foster common interfaces and to decrease the propensity for growth of inheritance trees.

2.6.2.2.5 Service Definition Option #5 GloMo Rooftop API based Networking API Service

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.1.5.

2.6.2.2.5.1 Description

This Networking API Service Definition approach is based upon the GloMo Rooftop API framework except that the primitives are specified in IDL. This is the Networking API that the MSRC proposed during JTRS Phase 1. Since this service definition includes inheritance, it can be considered a special instance of Alternative Option 4 Networking API Service Definition.

2.6.2.2.5.2 Architectural Considerations

See Service Definition Option # 4.

2.6.2.2.5.3 Relative Strengths and Weaknesses

See Service Definition Option #4. This option has the additional benefit of being specified in IDL.

2.6.3 Transfer Mechanism Analysis

2.6.3.1 Transfer Mechanism Criteria (Figures of Merit) and Ratings

(Note: the criteria numbers in the following tables refer to the detailed questions presented in section 3 of this document. For example, criteria 6.1 (Waveform Portability) is treated in sections 3.2.n.6.1, where n refers to Transfer Mechanism Option n (1-4).)

Numeric areas that are highlighted both at the top and lower level indicate areas where the Networking Working Group did not come to a consensus during the initial review of the analysis data. After leaving the meeting, the individual members considered the data and then met again to come up with a common view and position that is described in the SCA.

Table 5. Networking API Transfer Mechanism Figure Of Merit (FOM) Analysis

Criteria	Weight	Option 1: CORBA[4]	Option 2: CORBA[2] Non-CORBA[2]	Option 3: CORBA[3] CORBA-Ref	Option 4: CORBA[2] STREAMS[2]
6.1 Waveform Portability	20%	9	3	9	9
6.2 Ability to support legacy and future waveforms	15%	9	9	9	9
6.3 Performance (Efficiency/Latency)	15%	1	3	3	9
6.4 Extensible	5%	9	9	9	9
6.5 Dynamically construct/ deconstruct protocol stacks	10%	9	3	9	9
6.6 Component Interoperability (Component plug & play)	15%	9	9	9	3
6.7 Cost	10%	9	9	9	9
6.8 Commercial Acceptance	5%	1	1	1	3
6.9 Security	**binary (1 or 0)**	3	3	3	3
6.10 Ability to adapt to commercial trends	5%				
SCORE	100%	**6.95**	**5.45**	**7.25**	**7.35**

	Need to research with available data and re-assess.
Waveform Portability	Ability of waveform software/hardware to move from one JTRS platform to another without modification (other than a potential recompile).
Legacy support	Ability of the system to support existing waveforms in a consistent manner, i.e. that same manner as new waveforms are supported).
Performance	Ability of the system to support high speed, high data, low latency waveforms. Includes high data rate throughput and low command latency from red processor to modem cards.
Extensible	Ability of the architecture to be extended to support new waveform interfaces and mechanisms.
Dynamic stack construction	Ability of the design to support the dynamic (i.e. run time) build-up and tear-down of each layer of a protocol stack for each waveform.
Component interoperability	Ability of individual waveform components to be reused by other waveforms; the ability of components to support plug and play.
Cost	Protocol software entity development, maintenance, and porting cost (Higher score corresponds to lower cost.)
Commercial Acceptance	Degree to which the architecture is acceptable to the commercial world.

Security	Ability of the design to support the JTRS security architecture. This point is considered binary in that the alternate will either support or not support the JTRS security concept. As part of the evaluation of this criteria the ability of the alternate to support control/status bypass must be evaluated. Difficulty in implementing with the alternate is quantified under Cost.
Commercial Adaptability	The ability of the architecture to adapt to emerging commercial networking standards.

Ratings to Detailed Questions

The Combined Answers to the Detailed Questions are then Combined to yield a single 1, 3, or 9 value in the Combined Evaluation Factor

	Option 1: CORBA[4]	Option 2: CORBA[2] Non-CORBA[2]	Option 3: CORBA[3] CORBA-Ref	Option 4: CORBA[2] STREAMS[2]	
6.1.1 (6.1)	9	3	9	9	Note 1
6.1.2	9	9	9	9	
6.1.3	9	9	9	9	
6.2.1 (6.2)	9	9	9	9	
6.2.2	9	9	9	9	
6.2.3	9	9	9	9	
6.2.4	9	9	9	9	
6.3.1 (6.3)	1	9	3	9	Note 2
6.3.2	1	3	3	9	Note 3
6.4.1 (6.4)	9	9	9	9	
6.5.1 (6.5)	9	3	9	9	
6.5.2	9	9	9	3	
6.5.3	9	3	9	9	
6.6.1 (6.6)	9	9	9	9	
6.6.2	9	9	9	3	
6.7.1 (6.7)	9	9	9	9	
6.7.2	9	9	9	9	
6.7.3	9	9	9	9	
6.8.1 (6.8)	1	1	1	3	
6.8.2	1	1	1	1	
6.9.1 (6.9)	9	1	9	9	
6.9.2					
6.9.3	3	3	3	3	
6.10.1 (6.10)					

Note 1: Portability using Option #4 is worse (a 3) on the Networking API Service Definition if it is one of several mechanisms and better (a 9) if it is the only approved mechanism.

Note 2: Option #3 has the disadvantage of additional context switching over Options #2

Note 3: Option #1 has non deterministic behavior

2.6.3.2 Transfer Mechanism Options

The following sections provide summary information for the analysis that the NWG did for the Networking API Transfer Mechanism. The group initially derived five options for the transfer mechanism that cover the range of possible implementations. It was decided that the fifth option, API built around CF that allows any middleware, be removed because it is effectively building your own CORBA ESIO and security needs a consistent interface (i.e. GIOP). Also, the OE is abstracted out via the POSIX interface. The NWG then analyzed the remaining four options using questions developed from the evaluation described in Section 2.6.3.1. The ratings of answers provided the basis of the selection of the chosen option for the SCA.

2.6.3.2.1 Transfer Mechanism Option #1 CORBA, CORBA, CORBA, CORBA

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.2.1.

2.6.3.2.1.1 Description

This alternative uses CORBA for non-real-time setup of waveform and non-real-time setting of operation parameters, Waveform Control across Red-to-Black Boundary, real-time control, and Waveform Data Flow across the Red-to-Black Boundary.

2.6.3.2.1.2 Architectural Considerations

Protocol applications would be the same as any other application, there no special architectural requirements.

2.6.3.2.1.3 Relative Strengths and Weaknesses

This option has the following strengths:

1. Portability
2. Interoperability

This option has the following weaknesses:

1. Performance
2. Commercial acceptance

2.6.3.2.2 Transfer Mechanism Option #2 CORBA, CORBA, Non-CORBA, Non-CORBA

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.2.2.

2.6.3.2.2.1 Description

This alternate uses CORBA for real-time and non real-time control and data flow with the following exceptions: non-CORBA transfer mechanisms will be allowed for real-time waveform control across Red-to-Black Boundary, non-CORBA transfer mechanisms will be allowed for waveform data flow across red-to black boundaries and hiding non CORBA-compliant objects in a IDL wrapper/adapter (e.g. kernel protocols) . The fundamental concept of this option is to put adapters/wrappers around objects that do not fit well into the CORBA environment. An adapter converts CORBA calls to the local transfer methods of the wrapped/hidden device. For example when data flows across the red/black boundary (through the COMSEC device) there will be a CORBA endpoint on each side of the COMSEC device. The actual processing of the data

flowing through the COMSEC device will be done via another mechanism. However the adapter makes it seem as if the COMSEC device is a CORBA object. Also this method allows grouping large groups of protocols (e.g. TCP/IP in kernel space) into one CORBA object via an adapter.

2.6.3.2.2.2 Architectural Considerations

Protocol applications would be the same as any other application, no special architectural requirements, because the augmentations are hidden via the adapters.

2.6.3.2.2.3 Relative Strengths and Weaknesses

The strength of this option is the ability to port non CORBA software (e.g. legacy software) and hardware into the CORBA environment.

The weaknesses of this option are:

1. Portability may be an issue because interfaces are hidden via adapters; To port objects the entire wrapped object must be ported as a whole.
2. Performance could be affected by the addition of adapters and overhead associated with standard CORBA IOP.
3. Commercial acceptance.

2.6.3.2.3 Transfer Mechanism Option #3 CORBA, CORBA, CORBA, CORBA-Reference

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.2.3.

2.6.3.2.3.1 Description

This alternate is a superset of options 1 and 2.This option uses CORBA for real/non-real time control and data flow. Standard CORBA with IOP should be used unless there is a real time requirement that cannot be met. This option advocates that when there is a real time requirement that can not be met with the standard CORBA IOP, the standard CORBA IOP should be augmented. An example augmentation is to use CORBA pass by reference (e.g. shared memory transport and a shared RAM card). Also this alternate advocates using adapters/wrappers as described in option 2 as long as the objects are wrapped so that portability is not effected.

2.6.3.2.3.2 Architectural Considerations

Protocol applications would be the same as any other application, no special architectural requirements, because the augmentations are hidden via the CORBA interface.

2.6.3.2.3.3 Relative Strengths and Weaknesses

The strength of this option is the component interoperability, component portability, and processor independence. Every JTRS compliant box must support the CORBA interface and every application must adhere to the SCA Application Environment profile; therefore the protocols can be ported from platform to platform, and processor to processor the same as any other JTRS application. CORBA makes the location of an application transparent. This allows applications to be easily moved without effecting other applications which are dependent on them. Also this option specifies a method to address real-time issues associated with protocols without effecting the above specified advantages of CORBA. This option is a superset of the other options with the exception of the Option # 4 (e.g. STREAMS).

The weakness of this CORBA-based option relative to Option # 4 are performance and commercial acceptance. For example STREAMS modules/drivers would run in kernel space and

are at a higher priority than the other applications; therefore kernel space applications will run faster. Currently most protocols run in kernel space.

2.6.3.2.4 Transfer Mechanism Option #4 CORBA, CORBA, STREAMS, STREAMS

The following three sections summarize this option. For more detailed information regarding the analysis of this option refer to section 3.2.4.

2.6.3.2.4.1 Description

This option presents one example of a transfer mechanism other than CORBA for real-time data flow and control. An alternate transfer mechanism was investigated because wide band waveforms may require a more efficient transfer mechanism between network protocol layers than CORBA can provide. In addition this option examined a networking API transfer mechanism that is currently in use in the commercial world and can make maximum use of COTS networking protocol implementations.

The particular transfer mechanism investigated was UNIX System V STREAMS. The STREAMS environment can exist in operating systems with or without memory management. It abstracts the operating system from the protocol developer and provides a means to gain modularity and portability for protocols at the kernel level. In addition it provides the capability to dynamically build up and tear down a protocol stack which is a requirement for JTRS. The CF is used for instantiation and configuration of the protocol stack. CORBA based adapters present the set of non-real time control interfaces that are not hidden (i.e. used) by another STREAMS based protocol layer.

2.6.3.2.4.2 Architectural Considerations

The CF would have to provide the capability to load and unload kernel modules in a memory mapped OS because the STREAMS environment exists in kernel space.

The CF would have to provide the capability to push, pop, link and unlink STREAMS modules as part of its scheme for connecting protocol entities.

A rule set would have to be defined to accommodate the distribution of STREAMS modules across processors.

A rule set governing the use of CORBA based adapters with STREAMS modules would have to be defined.

2.6.3.2.4.3 Relative Strengths and Weaknesses

The strengths of the STREAMS option relative to the other transfer mechanism options are performance and commercial acceptance. The relative weakness is component interoperability (specifically interoperability between unlike processors). Because STREAMS has not been mandated then it becomes a weakness relative to a full CORBA implementation.

2.6.4 Additional Analysis Information

The following sections provide additional background information for the analyses that the Networking Working Group performed. This information was discussed within the WG as part of the overall analysis process and considered when determining the selected architecture.

2.6.4.1 Rationale for Alternate Transfer Mechanisms

2.6.4.1.1 Overview

This section provides the rationale for calling out alternate transfer mechanisms for real time data flow and control in the SCA. Real time data flow includes voice, video and data to be transmitted/received over the air. Real time control is the information that is attached to each packet of real time data for cases where per packet control information is required.

2.6.4.1.2 Wideband Waveform Requirements

2.6.4.1.2.1 Video/Voice/Data

The ORD and the JTRS Networking IPT Quick Look Report both state the need for video, voice and data capability. The video and voice will be packetized and will be multiplexed over the same waveform channel. Intended uses include collaborative planning (data conferencing, white-boarding, application sharing and file transfer), video teleconferencing (multiplexed audio, video and data), distance learning (streaming video and audio, file transfer and web page downloads), message and data exchange (SA distribution, fire control, imagery, etc).

2.6.4.1.2.2 Multicasting

Future tactical networks will make heavy use of multicasting to conserve bandwidth. For instance, Tactical C2, Situation Awareness, and video/audio streams will be multicast to multiple recipients.

2.6.4.1.2.3 Ad Hoc Networks

Out in the field, automatic formation and maintenance of networks will be required so that user services flow between platforms as they enter and leave LOS clusters without operator intervention.

2.6.4.1.3 Wideband QOS Considerations

2.6.4.1.3.1 Throughput

Based on the analysis in the JTRS Networking IPT (Government) Quick Look Report of future wide band waveform requirements, JTRS radios will need to support burst data rates in excess of 10 mbps on a single channel. Put multiple channels in a radio and the aggregate burst data rate will exceed 100 mbps. In addition, bandwidth requirements tend not to remain static. As TI users increase their dependence on digital battle command, monitoring, and communication applications their need for increased bandwidth will be analogous to user's unbounded needs in the commercial internet.

2.6.4.1.3.2 Latency

Real-time audio requires that latency not exceed ~ 200-250 ms for full duplex conversation to take place. Any greater latency and the participants in a conversation perceive it to be simplex. The latency figure includes the time from speaker enunciation to listener reception. This would include all hops through the net.

2.6.4.1.3.3 Isochronous Behavior

Video and audio transmission require isochronous behavior between source and destination. For streaming video and audio that is not interactive, buffering/queuing at the application level (i.e. end points) can ameliorate data rate fluctuations in the transmission path. When the audio/video

is interactive such as in a videoconference, then some of the burden to maintain an isochronous data stream shifts to the elements in the transmission path.

2.6.4.1.3.4 Resource Reservation

Real time audio/video will require transmission protocols that support QoS guarantees. One such protocol is Asynchronous Transfer Mode (ATM), which exists at OSI layer 2 (link layer). The protocol data unit (PDU) size for ATM is fixed at 53 bytes. If the transfer mechanism between the ATM layer and the Modem is not efficient the overhead of the transfer mechanism can become a significant portion of the data transfer between the ATM layer and the modem.

2.6.4.1.3.5 Multicast Filtering

Currently, the evaluation of multicast addresses occurs at the IP layer. In this case, traffic that is not of interest to a node in a network can still make its way into the IP layer where it will be dropped. This activity will consume processing resources in a radio. If the implementation's network stack & the transfer mechanism are not efficient, the consumption of valuable processing resources will be considerable.

2.6.4.1.4 Factors Affecting Performance

2.6.4.1.4.1 Data Copying

Each time a copy of a packet is made protocol stack performance is degraded. In commercial protocol stacks a buffer management scheme is used to minimize data copying. Existing ORBs copy information from client to server.

2.6.4.1.4.2 Context Switching

Context switching, both light and heavy weight, have an impact on performance because context switches take more time than a direct method invocation or function call. In a CORBA based application level protocol stack there will be context switching between the protocol layers. This is extra overhead that is not incurred in a kernel level protocol stack.

2.6.4.1.4.3 Boundary Crossings

Crossing the boundary from user space to kernel space and vice versa incurs a performance penalty. When transferring data between processes, ORBs currently make these transitions at both the client and server end.

2.6.4.1.4.4 Marshalling

CORBA ORBs will marshal data between a client and server in separate processes in separate address spaces. Some ORBs will even marshal between a client and server in the same address space. This is extra overhead.

2.6.4.1.4.5 GIOP

The General Inter-Orb Protocol (GIOP) defines the following constructs for a client to invoke operations on a server. The CORBA specification specifically states that GIOP must be implemented in all conforming ORBs but that internal ORB communication is not restricted to using GIOP. However, when communicating between separate processes, it is assumed that GIOP is used. The following tables were generated from the CORBA 2.3 specification and represent the contents of a RequestHeader that is used when invoking and operation on an object.

GIOP::RequestHeader_1_2 (structure)

Field	Size in Bytes or Composite Name
request_id	4
response_flags	1
reserved	3
target	TargetAddress
operation	1..n
service_context	IOP::ServiceContextList

GIOP::TargetAddress (union)

Field	Size in Bytes or Composite Name
object_key	1..n
profile	IOP::TaggedProfile
ior	IORAddressingInfo

GIOP:: IORAddressingInfo (structure)

Field	Size in Bytes or Composite Name
selected_profile_index	4
ior	IOP::IOR

IOP::ServiceContextList (sequence)

Field	Size in Bytes or Composite Name
	ServiceContext

IOP::ServiceContext (structure)

Field	Size in Bytes or Composite Name
context_id	4
context_data	1..n

IOP::TaggedProfile (structure)

Field	Size in Bytes or Composite Name
tag	4
profile_data	1..n

IOP::IOR (structure)

Field	Size in Bytes or Composite Name
type_id	1..n
profiles	sequence of TaggedProfile

Assuming the following:

- That target field in the RequestHeader is using an object_key as the TargetAddress

- The object_key contains a time stamp, POA ID and Object ID and the total size of the object_key is 36 bytes (17 character time stamp + "/thePOA/theServant" + null)
- That the operation name is three characters + null
- That the service_context is empty

This yields a request header size of 4+1+3+36+4 = 48 bytes. Added to a 53-byte ATM cell yields a packet of 101 bytes of which 47.5% is overhead. This calculation does not include any overhead for the CORBA transport itself. It can be seen that for small packets the CORBA overhead can surpass the size of the payload.

2.6.4.1.4.6 Processing Resources

Processing resources can very widely within JTRS radios. In a resource-constrained environment where size, weight and power limitations govern parts selection, efficient performance can become a necessity.

2.6.4.1.5 Leveraging COTS

The commercial world currently is standardized around kernel based protocol stacks. For extreme cases the protocols are embedded in hardware such as IP routers that support gigabit Ethernet. To take advantage of COTS software/hardware solutions for networking then alternate transfer mechanisms for protocols will need to be investigated.

2.6.4.1.6 Summary

In summary there are several factors that individually or in combination will drive an implementation of a wide band waveform towards an alternate transfer mechanism. Therefore the SCA specifically calls out alternate transfer mechanisms and prescribes a path for radio vendors to follow if the performance penalties incurred from the use of CORBA become too great.

2.6.4.2 Name – Value Pairs and the Service Definitions

Final definition of how to handle name –value pairs is a task that stills needs to be completed by the Networking Working Group.

2.6.4.3 Type Any

The CORBA Any type is required by Service Definition Option 1 to support multiple waveforms. Independent research of multiple ORBS has shown that the Any type is inefficient. This was a factor in picking Service Definition Option 3 over Service Definition Option 1. The following URLs point to research on ORB performance:

http://www.omex.ch/CorbaTB/corbatb.htm

http://www.beust.com/virginie/Benchmarks/

http://www.kav.cas.cz/~buble/corba/comp/

2.6.4.4 Example of CORBA/STREAMS Interoperation

2.6.4.4.1 Description

This section describes how STREAMS and CORBA based implementations of external protocol entities can be developed to interoperate. The example shown in this paper illustrates how a STREAMS-based stack on the Red Processor can interface with a CORBA-based red side link layer. The red-side link layer is assumed to communicate with a CORBA-based modem layer on the black side.

2.6.4.4.2 Waveform instantiation

Figure 6 shows a red side STREAMS based protocol stack after startup including relevant object representations. The Domain Manager (DoM), Red Processor Device (RP Dev) and Security Device (Sec Dev) and Object Request Broker (ORB) are shown. CORBA sits at the application layer and interfaces to the stack through the sockets API. The routing daemon which contains the routing protocols such as RIP, OSPF, DVMRP, etc also interfaces to the stack through the sockets API. At the bottom of the stack are three drivers, one for Ethernet, one for the INFOSEC and one pseudo-device. The Ethernet driver presents the Data Link Provider Interface (DLPI) interface to the IP module. The INFOSEC driver implements the interface to the INFOSEC. The INFOSEC driver does not present the DLPI interface itself because a link module may be pushed on top of it that is sending to and receiving from a modem module. Therefore the driver must be more generic. A separate DLPI module presents the DLPI interface to the IP module. This interface is the gateway to the Black Side subnet. The pseudo-device is for moving network functionality up into application space. As shown on the diagram, external IIOP traffic flows through the Ethernet and up the protocol stack to the ORB or gets routed by the IP layer to the INFOSEC interface for transport to a Black Processor. Since there are no waveforms running, there is no real time data flow.

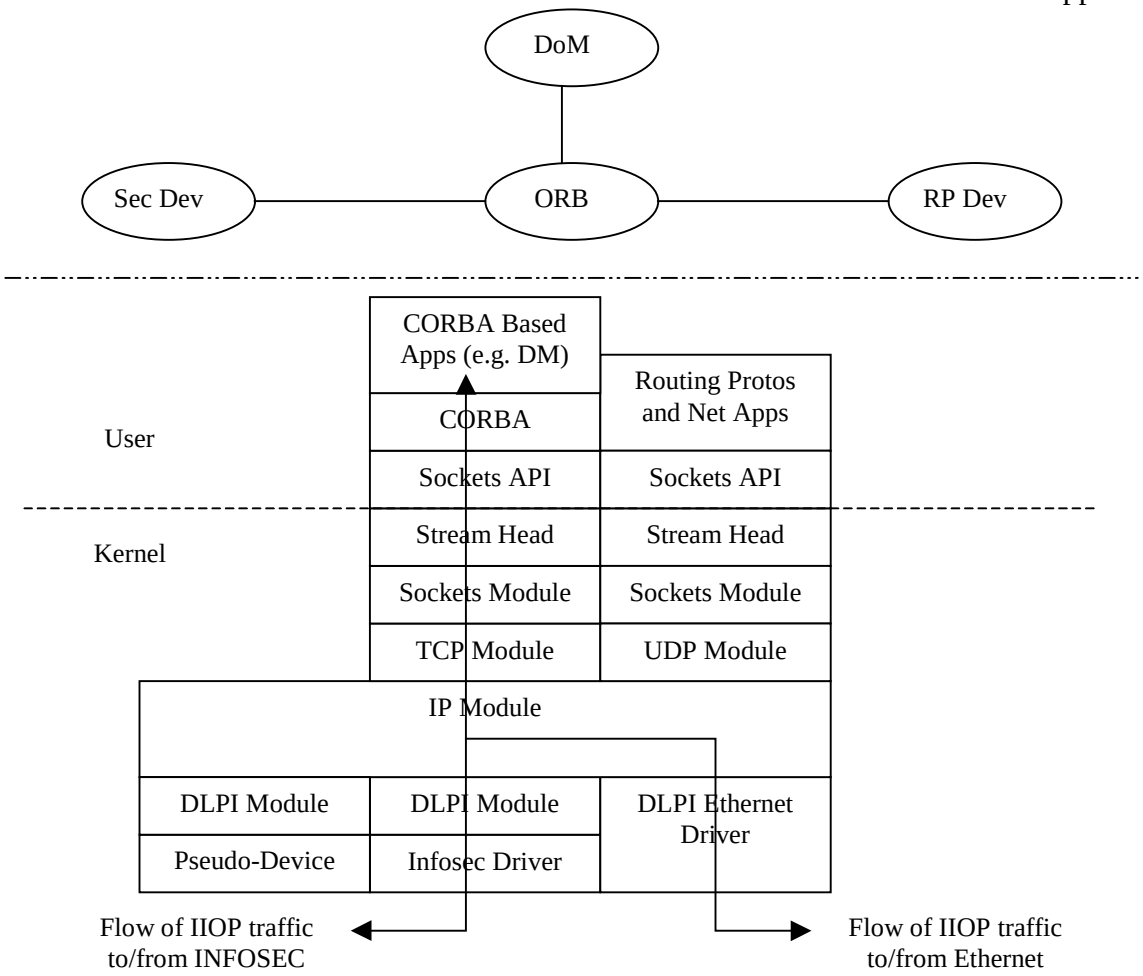


Figure 6 - Red Side (Single Processor) Stack After Startup

Figure 7 shows the stack at the beginning of waveform instantiation. The RP Dev has been commanded by the Domain Manager to load and execute the application software. Upon execution the factories are created. The Domain Manager has then commanded a factory to create a link adapter that opens a STREAM to the pseudo device.

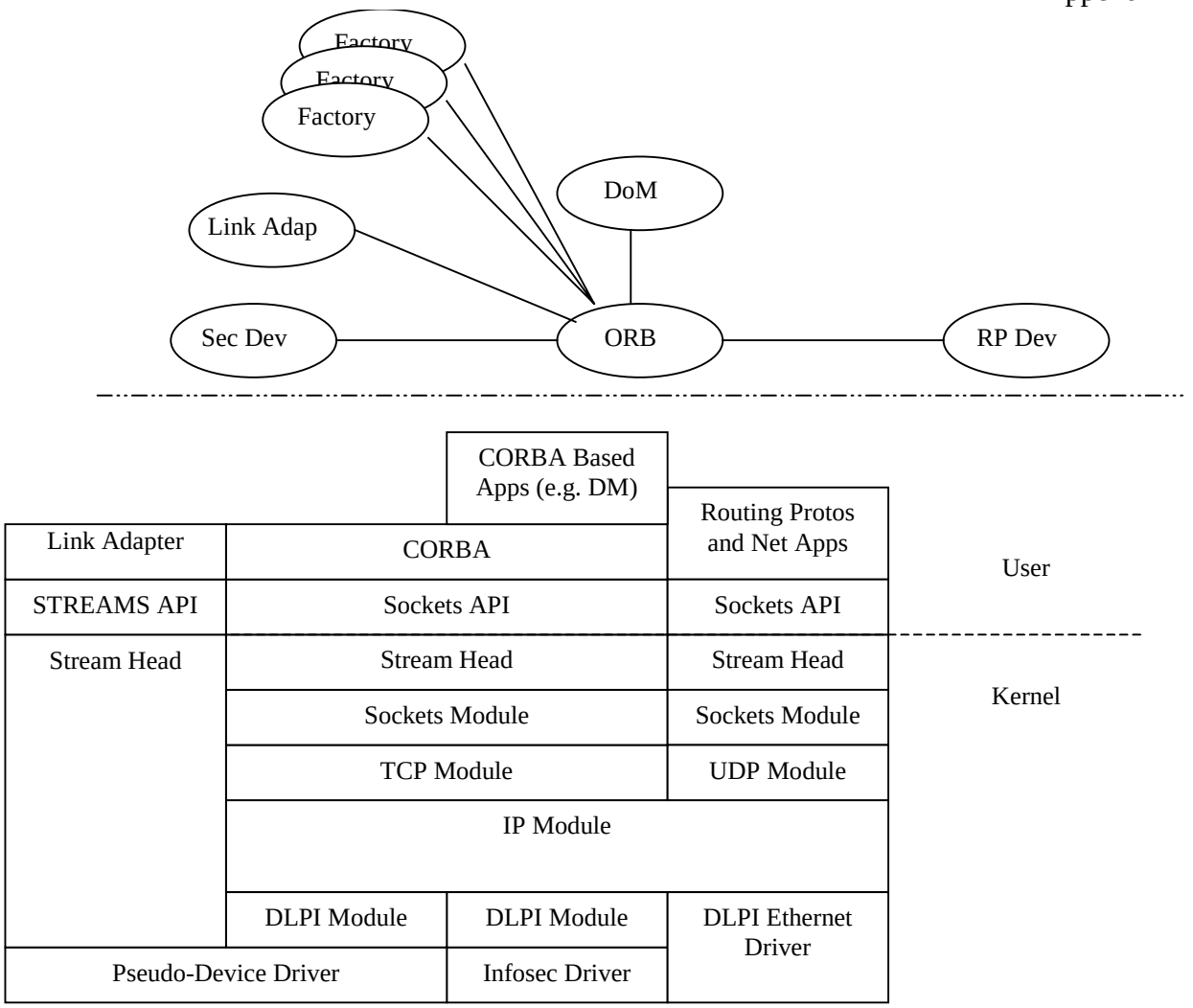


Figure 7 Red Side Stack Waveform Instantiation Step 1

Figure 8 shows the next step in the instantiation process. The Domain Manager commands another factory to create the Red Link Resource and a factory on the black side to create a Modem Resource (not shown). The Domain Manager then connects the Link Adapter to the Red Link Resource that is in turn connected with the modem resource on the black side. In addition an IP address is assigned to the pseudo-device driver interface. The interface can now be configured as an IP routing gateway. Assuming that the waveform has been fully instantiated and configured on the black side as well, the Radio System is ready to receive and transmit networked traffic over the air. The arrowhead terminated line illustrates the path that the on-air traffic would take through the STREAMS stack, CORBA, the link adapter and the Red Link Resource.

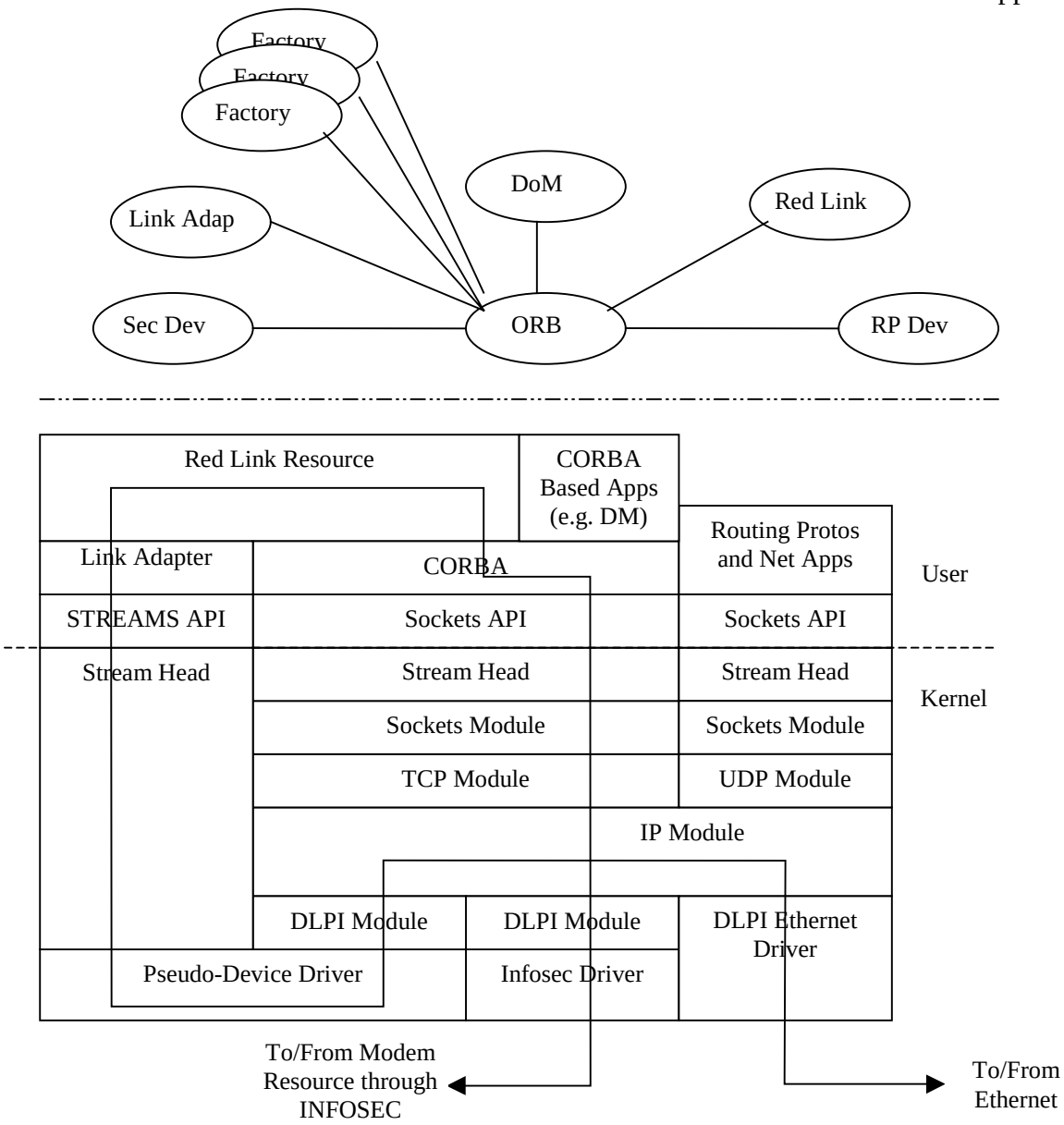


Figure 8 Red Side Stack Waveform Instantiation Step 2

3 SUPPLEMENTAL DATA ON OPTIONS CONSIDERED

This section contains additional information on the five Networking API Service Definition options (section 3.1) and the four Networking API Transfer Mechanism options (section 3.2).

3.1 NETWORKING API SERVICE DEFINITION OPTIONS

3.1.1 Option 1 – Small Networking API Service Definition Using Name/Value Pairs

Interface Type: Networking API Service Definition

Name: Option 1 – Small Networking API Service Definition Using Name/Value Pairs

Date: 17 June 2000

Revision: 4.0

Author: Jim Stevens (jasteven@collins.rockwell.com)

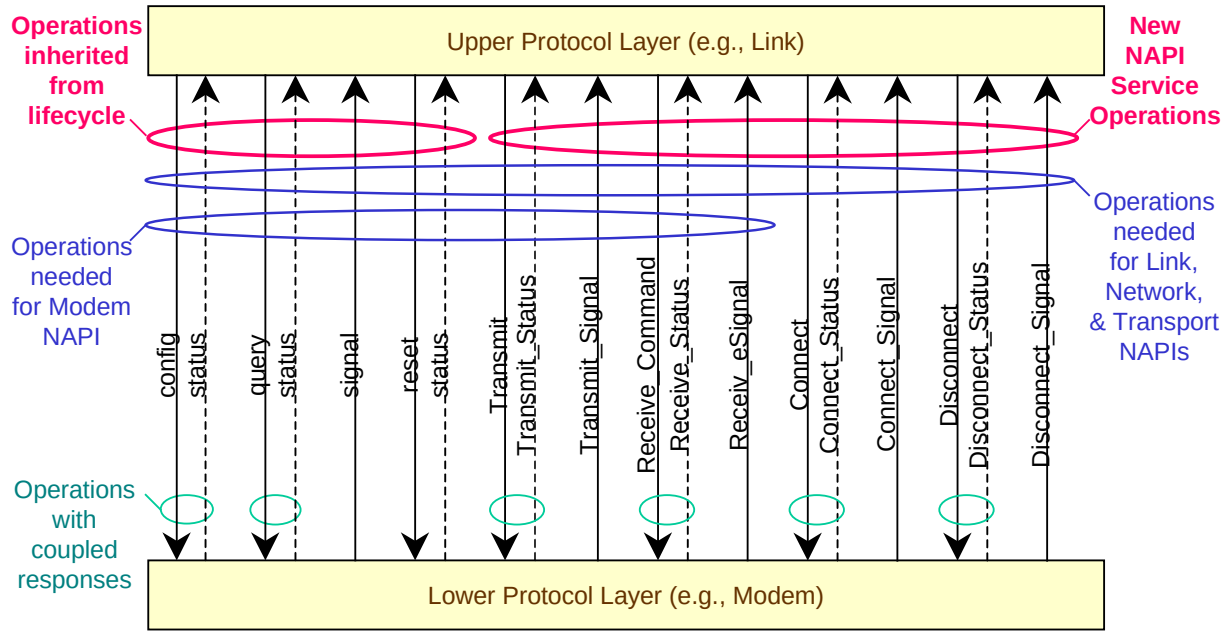
Company: Rockwell Collins

3.1.1.1 Description

This Networking API option uses a small set of operations (config, query, response, transmit, receive, etc) along with name/value pairs to pass control information. This Networking API option is based upon the output of the MSRC Networking IPT meeting held at Fort Wayne, Indiana on 18-21 January 2000.

3.1.1.2 Diagram

The following diagram shows the operations defined by this Networking API Service Definition:



3.1.1.3 Theory of Operation

This Networking API Service Definition option defines a small set of possible operations along with the specification of the control and data within the operations as well as a method of interface configuration control, interface version identification, and interface negotiation.

This option provides a small service definition since it defines a fixed set of operations that all Networking API Service Definitions could draw from. In particular, the Modem Networking APIs could draw from a set of eight possible operations while the Link, Network, and Transport Networking APIs could draw from a set of twelve possible operations. A small number of operations helps to develop consistency between Networking API Service Definitions and facilitates going from one waveform's service definition to another.

This small number of operations was developed by the group based upon our experience with a large number of waveforms, including most of the waveforms called out by the JTRS ORD as well as a variety of other commercial, military, and emerging waveforms (such as IEEE 802.11, VRC-99, and Soldier Phone). We were able to merge all of the operations needed for service interfaces for this plethora of waveforms down to the small number of operations defined. Also, some of the operations can be reused from the Core Framework Lifecycle Interface thus reducing the development effort and possibly the implementation footprint required.

This option specifies that control information is passed by name/value pairs specified in IDL while the traffic data payload could be passed by value (more portable approach) or by reference (better performance approach). Whether the traffic payload is passed by value or by reference will be transparent to the protocol resources. If the resources are in different memory spaces, then the traffic data buffer is passed by value (the same way used in IDL). However, if the resources are in the same memory space, then the traffic data buffers may be passed by reference, thus decreasing latency from the buffer copying. The key is to define the semantics of the operations so that the operation looks identical to the software whether buffers are passed by value or by reference.

A potential refinement to this Networking API is to define buffer definitions and operations for efficient protocol layering. Simple buffers (basically just a block of data) would be used for simple protocols like push-to-talk voice while more complex buffer definitions (similar to socket and STREAMs m_bufs) could be used for more complex network protocols (for efficient layering and packet header manipulations up and down the protocol stacks).

This option defines a method of interface configuration control, interface version identification, and interface negotiation. It was designed to give vendors and users great freedom and flexibility in trading off performance, portability and cost drivers. It promotes open interface standards for waveforms and allows customers to assess the impact of changes. Its configuration control provides mechanism for managing evolution and tracking changes. The flexible service definition as well as the configuration control approach was modeled after that used to specify and manage MIBs for SNMP and CMIP.

A Networking API Service Definition will be defined at the functional level for a waveform or set of waveforms. The developer of a waveform implementation will publish the Networking API Service Definition(s) of its visible network components. The service definition will specify what subset of the twelve operations are used along with the name/value pairs (including the value data structures and enumeration values). Each Networking API Service Definition will be assigned a globally unique identifier (GUID) that:

- Specifies the defined Networking API service interface
- Specifies the version number of the interface
- Must be visible to the Domain Manager profile.

There will be a controlling body for GUIDs that controls the control assignment of GUIDs. This configuration control body should be an open standards body similar to the IETF or the IEEE. This controlling body would assign a status for each Networking API Service Definition similar to the way that the IETF assigns statuses to Internet standards today. Potential statuses could include:

- mandatory for a particular waveform or set of waveforms (i.e. if a vendor implements one of the particular waveforms, then they must implement this Networking API Service Definition to be JTRS compliant)
- recommended for a particular waveform or set of waveforms (i.e. if a vendor implements one of the particular waveforms, then it is recommended, but not required, that they implement this Networking API Service Definition to be JTRS compliant)
- allowable for a particular waveform or set of waveforms (i.e. the vendor is allowed, but not required to implement this Networking API Service Definition to be JTRS for one of the particular waveforms)
- Other possible statuses including experimental, not recommended, etc.

Mandatory and/or recommended Networking API Service Definition(s) would be identified and published for each waveform that is standardized for JTRS. Companies and other organizations are free to request their own GUID for their own Networking API definitions. The companies and organizations can then decide whether to make their own interfaces definitions and standard Networking API extensions public or private. Thus, this interface is extensible to support growth and customization.

Any Networking API Service Definitions that are chosen as JTRS standards (i.e., mandatory or recommended) must be published and non-proprietary. However, vendors can choose to keep the details of particular Networking APIs private or proprietary (so that the status is likely to be allowable or some other.) The owner of interface controls version numbering. This is similar to the approach used to managing MIBs in SNMP and CMIP.

For example, suppose that there is a JTRS mandatory Networking API Service Definition for the SINCGARS waveform. If a vendor develops a new proprietary way to mitigate co-site interference that requires a new SINCGARS Networking API Service Definition, then that vendor could get a GUID for its new Networking API Service Definition. The vendor would have the freedom to keep the details of the new Networking API Service Definition proprietary or to make it open. However, the new Networking API Service Definition could only become mandatory or recommended if the vendor made the Networking API Service Definition open. In either case, the vendor could now develop and sell a SINCGARS modem object that supports both the mandatory Networking API Service Definition and this new allowable Networking API Service Definition. Since there is a GUID, the Domain Manager can determine at instantiation whether the SINCGARS modem object is being linked together with other objects that know how to use the allowable SINCGARS service definition. If other such objects were available, then a better co-site mitigation JTRS radio would result. If the other objects only knew how to

support the mandatory Networking API Service Definition, then an operational JTRS radio would result that runs the SINCGARS waveform.

All visible interfaces will be uniquely identified. A mechanism will be provided within the Domain/Resource Manager to identify the interfaces used by the resources. When the Domain/Resource Manager instantiates a Resource to satisfy a User request it selects the resource with matching interface.

The twelve standard operations are:

1. Config (param_name_1, param_value_1, …)

The Config operation is used by upper layer protocols to set persistent parameters in lower layer protocols. For example, it could be used in a Modem Networking API Service Definition to set frequency, hop_set, mode (AM, FM), etc. The Config operation could have an optional associated Status response. The associated Status response format is Status (GUID, status_name_1, status_value_1, …).

2. Query (param_name_1, …)

The Query operation is used by upper layer protocols to request values of persistent parameters or status from lower layer protocols. For example, it could be used in a Modem Networking API Service Definition to request Config_parameters, Modem state (receiving, idle), queue_size, etc. The Query operation has an associated Status response. The associated Status response format is Status (GUID, status_name_1, status_value_1, …).

3. Reset ()

The Reset operation is used by upper layer protocols to set persistent parameters in lower layer protocols to known default status. The Reset operation is similar to a power on/power off sequence. The associated Status response is the same format is Status (GUID, status_name_1, status_value_1, …). (Note that this command is not currently in the Core Framework Lifecycle Interface, although it looks as if some combination of the Lifecycle Stop, Config, Initialize operations provide the same behavior. The exact combination is TBD.)

4. Signal (signal_name_1, signal_value_1, …)

The Signal operation is used by lower layer protocols to provide an asynchronous signal or status to upper layer protocols. For example, it could be used in a Modem Networking API Service Definition to indicate synch_detect, TDMA_frame_start, alarm, etc. (Note that the Signal operation was been added to the Core Framework Lifecycle Interface in SCA version 0.3.)

5. Transmit (persistent_set_id, <transient values>, buffer_handle)

The Transmit operation is used by upper layer protocols to tell lower layer protocols to transmit a buffer of data. The persistent_set_id identifies a set of already loaded parameters (such as frequency channel or hop set). The <transient values> are zero or more parameters (such as transmit power or TDMA transmit slot) only used for this transmission and are passed as name/value pairs (although an alternative approach which would be slightly more efficient would be to pass using IDL). The Transmit operation could have an optional associated Transmit_Status response. The Transmit_Status format is identical to the Transmit_Signal operation.

6. Transmit_Signal (<statuses>, buffer_handle)

The Transmit_Signal operation is used by lower layer protocols to return the status associated with an attempted transmission to an upper layer protocol. The <status> are one or more status associated with this transmission. Example <status> for a Modem Networking API Service Definition could be good, failed due to overflow, CSMA timeout, etc. The <status> are passed by name/value pairs (although an alternative approach which would be slightly more efficient would be to pass using IDL). The buffer_handle may be empty.

7. Receive_Command (persistent_set_id, <transient values>, buffer_handle)

The Receive_Command operation is used by higher layer protocols to tell lower layer protocols to receive a buffer of data. The persistent_set_id points to a set of already loaded parameters. An example persistent set of parameters for a Modem Networking API Service Definition could include frequency channel, hop set, etc. The <transient values> are zero or more parameters (such as TDMA receive slot for a Modem Networking API Service Definition) that are only used for this reception. The buffer_handle may be empty. The Receive_Command could have an optional associated Receive response. The Receive response is identical to the Receive_Signal operation.

8. Receive_Signal ( )

The Receive_Signal operation is used by lower layer protocols to tell upper layer protocols about the status a Receive command. The <status> are one or more status associated with this reception. An example set of <status> for a Modem Networking API Service Definition could include good_reception, CRC_error, corrected_errors, Signal_to_Noise, etc. The <status> are passed by name/value pairs (although an alternative approach which would be slightly more efficient would be to pass using IDL). The buffer_handle may be empty (in the case of an incomplete or errored reception).

9. Connect (persistent_set_id, <transient values>)

The Connect operation is used by upper layer protocols to tell lower layer protocols to set up a connection with its peer protocol entity in a different radio. The persistent_set_id identifies a set of already loaded parameters (such as frequency channel or hop set). The <transient values> are zero or more parameters (such as address or quality of service) only used for this connection and are passed as name/value pairs (although an alternative approach that would be slightly more efficient would be to pass using IDL). The Connect operation could have an optional associated Connect_Status response. The Connect_Status format is identical to the Connect_Signal operation.

10. Connect_Signal (<statuses>)

The Connect_Signal operation is used by lower layer protocols to return the status associated with an attempted Connect operation to an upper layer protocol. The <status> are one or more status associated with this connection attempt. Example <status> for a Link Networking API Service Definition could be connection address, good, failed due to address not found, etc. The <status> are passed by name/value pairs (although an alternative approach which would be slightly more efficient would be to pass using IDL).

11. Disconnect (persistent_set_id, <transient values>)

The Disconnect operation is used by higher layer protocols to tell lower layer protocols to tear down a connection with its peer protocol in a different radio. The persistent_set_id points to a

set of already loaded parameters. An example persistent set of parameters for a Link Networking API Service Definition could include frequency channel, hop set, etc. The <transient values> are zero or more parameters (such as address for a Link Networking API Service Definition) that are only used for this disconnection. The Disconnect could have an optional associated Disconnect_Status. The Disconnect_Status response is identical to the Disconnect_Signal operation.

12. Disconnect_Signal (<status>)

The Disconnect_Signal operation is used by lower layer protocols to tell upper layer protocols about the status associated with a Disconnect operation. The <status> are one or more status associated with this reception. An example set of <status> for a Link Networking API Service Definition could include address, confirmed disconnect, unconfirmed disconnect, etc. The <status> are passed by name/value pairs (although an alternative approach which would be slightly more efficient would be to pass using IDL).

The first eight operations above are sufficient for a Modem Networking API Service Definition for every waveform our working group knows about, including all of the waveforms identified by the JTRS ORD. (For example simple push-to-talk waveforms may only use the Transmit and Receive_Signal operations while more sophisticated networking waveforms may use all or almost all of the operations.) The last four operations above may also be required for Link, Network, and Transport Networking API Service Definitions.

3.1.1.4 Operational Environment (OE)

Note that with the addition of a Reset operation to the current Lifecycle interface, the Config, Query, Signal, and Reset operations could be support by the Lifecycle interface. The Transmit, Transmit_Signal, Receive_Command, Receive_Signal, Connect, Connect_Signal, Disconnect, and Disconnect_Signal operations are a new interface outside the current core framework.

3.1.1.5 Protocol Layer Applicability

The first eight operations above are sufficient for a Modem Networking API Service Definitions for every waveform our working group knows about, including all of the waveforms identified by the JTRS ORD. (For example simple push-to-talk waveforms may only use the Transmit and Receive_Signal operations while more sophisticated networking waveforms may use all or almost all of the operations.) All twelve operations are sufficient for Link, Network, and/or Transport Networking API Service Definitions for every waveform examined.

3.1.1.6 Networking API FOM Analysis

3.1.1.6.1 Waveform Portability

Does the Networking API Service Definition approach foster waveform portability?

Yes, through the use of a small common set of operations and a consistent way of passing control and data. Also, the use of name/value pairs improves Human Machine Interface (HMI) portability because generic HMI software can support new Networking APIs through the incorporation of a table of the new name/value pairs. [This means that the Small Networking API Service Definition can support new Networking APIs similar to how commercial Simple Network Management Protocol (SNMP) managers can support new SNMP Management

Information Base (MIB) definitions through the incorporation of the new MIB data definition without having to write new SNMP manager software. SNMP MIBs currently defined as name values using a variant of the ISO Abstract Syntax Notation One (ASN.1).]

Note: It is assumed that 'waveform portability' means porting a complete waveform (and no hardware) from one JTRS platform to another, and not mixing and matching waveform components.

3.1.1.6.1.1 Identify the porting steps required to take a waveform to another JTRS Platform and evaluate the impact that the Networking API Service Definition approach has on that process.

The following table lists the projected porting steps for a wide range of porting stimulus.

Porting Stimulus	**Description**	***Projected Porting Steps Required Because of the Networking API Service Definition***
Environment Change: Different Processor	Waveform must be ported to a different processor	*No Networking API effect*
Environment Change: Different OS	Waveform must be moved to a different OS	*No Networking API effect*
Environment Change: Different Programming Language	Waveform Objects must interact with objects on the new platform that are written in a different language	*No Networking API effect*
Environment Change: Different Platform Hardware	Waveform is hosted on new platform hardware that may require additional or different control, e.g. modem, INFSOEC, red I/O	*No Networking API effect*
Environment Change: Different Transport Mechanism	Waveform is hosted on a new platform that uses a different transport mechanism than the source platform	Direct Rehost **or** Translation Shim **depending on the transport mechanism.**

The projected porting steps are defined from least to most effort in the following table.

Porting Step	**Description**	**Effort Level**
Direct Rehost	Same processors, OSs etc. – No changes required	None
Recompile	Recompile of those portions of the waveform that are hosted on a different processor	Minimal
Translation Shim	Does the Networking API approach require that shims be added to the waveform to port to the new platform?	Moderate
Edit/ReWrite	Does the Networking API approach require such drastic code changes that edit/rewrite is necessary?	Moderate-Extensive

3.1.1.6.1.2 Does the Networking API Service Definition approach foster waveform portability for software that already supports an existing interface, e.g. ARINC 732?

Yes, through Translation Shims for a new Networking API. It would be possible to create a Networking API using name/value pairs that maps into the functionality of an existing interface. Then a translation shim is all that would be required to go from the Networking API to the existing interface.

Note, however, that as Networking APIs become standardized for waveforms, it may be more difficult to support a plethora of existing interfaces for that waveform. For example, some interfaces change semantics and operations depending upon the state of the waveform. If the standard Networking API was not designed to that type of state semantics, then the Translation Shims become more difficult so that eventually it becomes easier to just edit or rewrite the application than it is to use the translation shim.

3.1.1.6.1.3 What is the impact of the Networking API Service Definition approach on the current SCA definition? And does it affect portability?

1. Does the Networking API approach change the definition of any existing interfaces? *Depends upon the outcome of whether the SCA should add a Reset operation or whether this Networking API Service Definition approach should use some combination of Stop, Config, and Initialize.*

2. How many new interfaces are created by the Networking API approach? *Only one new interface consisting of eight new operations is required for this Networking API approach. Note that this new interface would be defined "outside" the Core Framework so that this Networking API approach does not impact the Core Framework*

3. What is the impact of the Networking API approach on the Core Framework? *The Core Framework will have to be able to track Networking API Service Definition GUIDs and version numbers in resource profiles so that it can instantiate resources that can interoperate.*

3.1.1.6.2 Able to support new networking and non-networking waveforms

Yes, by modifying already developed Networking API Service Definitions and changing the version number of creating a new Networking API Service Definition and getting a new GUID.

3.1.1.6.2.1 Does the Networking API Service Definition approach allow incorporation of extensions for new functionality?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which operations are used and what the name/value pairs are.

3.1.1.6.2.2 Does the Networking API Service Definition approach support future waveforms in a consistent manner (e.g. reuse of the Networking API for multiple future waveforms)?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which operations are used and what the name/value pairs are.

3.1.1.6.2.3 Does the Networking API Service Definition approach support creation of Networking APIs that can be used by networking and non-networking waveforms?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which operations are used and what the name/value pairs are.

3.1.1.6.2.4 Is there a projected waveform that cannot be supported by the Networking API Service Definition approach?

No, since the twelve operations specified by Networking API Service Definition approach support operations needed by waveforms at the Modem through Transport protocol layers and the implementer has the freedom and flexibility to define their control and traffic parameters as needed.

3.1.1.6.3 Able to support legacy networking and non-networking waveforms

3.1.1.6.3.1 Does the Networking API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking API?

This service definition approach has more overhead for sending control (due to the parsing of the name/value pairs) than an approach that passes parameters is specific IDL without any name/value pairs. Note that this service definition approach could provide significantly lower overhead operation operating with a transfer mechanism that supports passing by reference as compared to a transfer mechanism that enforces pass by value.

3.1.1.6.3.2 Does the Networking API Service Definition approach allow incorporation of extensions for new functionality?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which operations are used and what the name/value pairs are.

3.1.1.6.3.3 Does the Networking API Service Definition approach support legacy networking waveforms in a consistent manner (e.g. reuse of the Networking API for multiple legacy waveforms)?

The Networking API provides a consistent set of operations (Config, Query, etc) upon which to standardize the Networking API definitions for a particular waveform. Thus, it provides consistency in operations.

The Networking API allows great flexibility in defining the name/value pairs and deciding which operations to support for a particular instantiation of the Networking API for a given waveform or waveforms. The actual instantiation is identified by the globally unique ID. Whether these name/value pairs are consistent across multiply waveforms is a function of the designers of the Networking API instantiations and the configuration control board that decides which Networking API instantiations should be selected as standards.

3.1.1.6.3.4 Does the Networking API Service Definition approach support creation of Networking APIs that can be used by networking and non-networking waveforms?

Yes, the set of operations support both and name/value pairs can be defined to support both.

3.1.1.6.3.5 Is there a known waveform that cannot be supported by the Networking API Service Definition approach?

No, and the IPT considered the operations required by the JTRS legacy waveforms.

3.1.1.6.4 Performance (Efficiency/Latency)

3.1.1.6.4.1 Does the Networking API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking API?

This service definition approach has more overhead for sending control (due to the parsing of the name/value pairs) than an approach that passes parameters is specific IDL without any name/value pairs. Note that this service definition approach could provide significantly lower overhead operation operating with a transfer mechanism that supports passing by reference as compared to a transfer mechanism that enforces pass by value.

Note that Name/Value pairs are passed in CORBA IDL using the "Any" data type. Any's are significantly slower (often having latencies twice as large as other data types such as Short, Long, or Records) according to data from the following published research:

DII COE Real-Time ORB Trade Study Release

Performed by Boeing for ESC/PAM

Contract Number F34601-96-C-0720 98-ESC-13

Document Number D204-31159-1 ORIG

http://www.ois.com/technical/rt_diicoe/release.html

Research on Real-time CORBA

Electrical and Computer Engineering, University of California, Irvine

http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html

Thus, the use of Name/Value pairs above CORBA as the Transfer Mechanism will have significantly slower performance than the use of other service definitions above CORBA.

3.1.1.6.4.2 Does the Networking API Service Definition approach support high data throughput, low latency control?

Control using this Networking API approach has a small additional latency of the name/value pair look-up as compared to an approach using IDL attributes.

3.1.1.6.4.3 How is performance affected by any required message parsing within the producer and consumer resources?

Performance is less efficient using this approach as compared to a "pure IDL" approach due to the additional parsing required by the name/value pairs.

3.1.1.6.5 Extensibility

3.1.1.6.5.1 Does the Networking API Service Definition approach allow Networking API definition to be extended to support new waveform interfaces, functionality, representation, and mechanisms?

Yes, by modifying already developed Networking API Service Definitions and changing the version number of creating a new Networking API Service Definition and getting a new GUID.

3.1.1.6.5.2 Does a mechanism(s) exist to limit proliferation of published Networking API extensions?

This Networking API approach per se does not have any mechanisms to limit proliferation of published Networking API extensions. However, this approach assumes that there is a standard configuration control body that determines which published Networking API extensions are the "standard" Networking API for a given waveform. In that sense it is very similar to SNMP which will support a plethora of MIB definitions, but has a limited set of MIBs defined by the IETF as mandatory.

3.1.1.6.6 Networking API Reuse at same layer

3.1.1.6.6.1 Does the Networking API Service Definition approach foster Networking API reuse at the same layer, e.g. use the same modem Networking API definition for multiple (different) waveforms?

Although this approach has a small number of operations, this approach does not prohibit but does not foster reuse since it does not mandate common name/value pairs. The implementer is allowed, but not required to provide such reuse. Also, the configuration control body could

foster such reuse through what Networking API Service Definitions it chooses to label as mandatory, recommended, or other.

3.1.1.6.6.2 Does the Networking API Service Definition approach foster commonality of interface?

1. Common Operation Set? *Yes through the small set of operations.*

2. Common Parameter Set? *No (although the standard configuration control body could foster such commonality).*

3. Common Message/Data Format? *Partially yes because it does have a common operation format, although the actual format of the name/value pairs could be different (i.e. have different value structures).*

3.1.1.6.7 Component Interoperability (within the box, component plug and play)

3.1.1.6.7.1 Does the Networking API Service Definition approach foster development of Networking APIs that allow individual waveform components to be reused by other waveforms?

No, the Networking API itself does not per se, but the concept of a standard configuration body coupled with the globally unique IDs does foster such development.

3.1.1.6.7.2 Do the Networking API definition, implementation, and configuration control methods support component interoperability?

1. Does the Networking API approach support interface negotiation (e.g. Global User ID)? *Yes*

2. Does the Networking API approach support interface version query? *Yes*

3. Does the Networking API approach support backward interface compatibility? *Yes, through GUID & version identification. In order to support backward compatibility, the waveform needs to support previous versions of the same Networking API Service Definition or multiple different Networking API Service Definitions for the same waveform.*

3.1.1.6.8 Cost

3.1.1.6.8.1 Is the Networking API Service Definition approach a cost driver?

1. Implementation Cost? *Moderately higher than "pure IDL" due to second layer of name/value pair representations, but much less than allowing GUID only and any functional & representation definition as long as it is documented and identified by the GUID. (Should be similar to that required to implement SNMP & SNMP MIBs. The cost could significantly drop with the introduction of name/value pair compilers and development tools similar to those used by SNMP.) Note that the actual cost will vary depending upon the number of parameters and their complexity. This similar to how SNMP MIB implementation costs vary based upon the number and type of parameters supported.*

2. Maintenance Cost? *Moderately higher than "pure IDL" due to second layer of name/value pair representations, but much less than allowing GUID only and any functional & representation definition as long as it is documented and identified by the GUID. (Should be similar to that required to maintain SNMP & SNMP MIBs. The cost could significantly drop with the introduction of name/value pair compilers and development tools similar to those used by SNMP.)*

3. Porting Cost? *If talking with applications that already understand/support Networking API, then it should be the same to slightly higher than "pure IDL" due to second layer of name/value pair representations. (Similar to how inexpensive it is to port SNMP applications & agents.)*

3.1.1.6.9 Commercial Acceptance

3.1.1.6.9.1 Does the Networking API Service Definition approach have current or projected commercial acceptance, (e.g. SDRF, IEEE, OMG, or de facto)? (No, some, or widespread commercial acceptance?)

Many wireless standards are being developed due to the rapid explosion in commercial wireless products. Although we do not know which of these commercial initiatives will achieve commercial de facto standard acceptance similar to the Hayes modem AT command set, several such interface de facto standards are bound to arise. The one that wins is probably going to be based around getting initial market penetration in both the commercial cellular data market, the wireless LAN market, and the consumer home/personal network market. Since this Networking API is being defined for JTRS which is focused on military applications, it is not likely that this Networking API (nor any Networking APIs) will achieve widespread commercial acceptance. Although it is likely that any JTRS Networking API will achieve some commercial acceptance (assuming there are no technical problems with the Networking API). Since the JTRS addresses a large, important military market (although small in comparison to the commercial market), any JTRS Networking API that does not have technical problems is likely to be accepted by the SDRF and/or the IEEE.

Since, this Networking API approach has similarities to SNMP, which has widespread commercial acceptance and also can use the IDL transport mechanisms, so should be no technical reasons that the commercial world would not accept it.

Thus, the combination of the exploding commercial wireless market coupled with the fact that this Networking API approach has not technical negatives means that this approach, if selected as the JTRS Networking API, would likely achieve some commercial acceptance and become a SDRF and/or IEEE standard.

3.1.1.6.9.2 Does the Networking API Service Definition approach foster the adoption of Networking APIs with current or projected commercial acceptance, (e.g. DLPI, Sockets, de facto (Microsoft))? (No, some, or widespread commercial acceptance?)

No. This Networking API approach has similarities to SNMP but is different. It will allow Networking API instantiations that are functionally similar to some commercial standards, but the representation will be different.

3.1.1.6.10 Security

This section requires further work and coordination with the Security IPT.

3.1.1.6.10.1 Do the JTRS security requirements prevent the use of this Networking API definition or implementation?

1. Use of IDL Networking API definition if red and black objects can only communicate through INFOSEC adapters? *This is a question if all messages flowing from red to black must pass through the generic message interface of an INFOSEC adapter. How can IDL interface definitions be used between red and black objects?*

A complete answer will await meeting with the Security IPT.

3.1.1.6.11 Ability to Adapt to Commercial Trends

3.1.1.6.11.1 Is the Networking API Service Definition approach able to adapt to emerging commercial networking standards?

This Networking API Service Definition approach will be able to adapt functionally to emerging commercial standards through its use of the small list of operations and a very flexible name/value pair definition. However, this approach will be unlikely to adapt and reuse emerging commercial standards as is.

3.1.2 Option 2 – Super Networking API

Interface Type: Networking API Service Definition

Name: Option 2 - Super Networking API

Date: 31 MAR 00

Revision: 5.0

Author: M. Di Mare

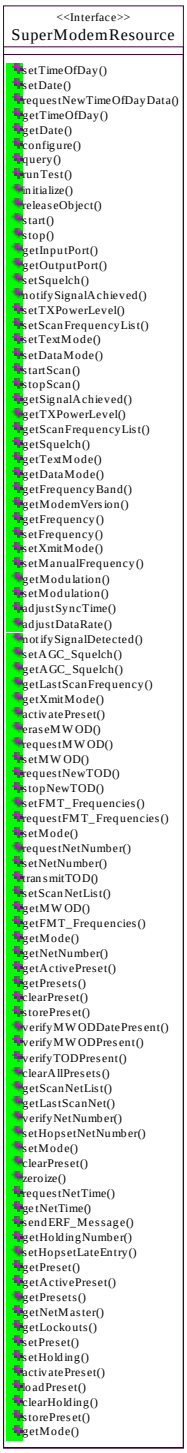
Company: BAE SYSTEMS

3.1.2.1 Description

The Super Networking API is a term used to describe the goal of defining a single universal interface for all waveform applications. Each level; Network, Link, and Modem would have its own Super Networking API and all JTRS compliant applications would use these Networking APIs when crossing these OSI boundaries.

3.1.2.2 Diagram

The Super Networking API approach can best be illustrated by providing an example. The following diagram shows a preliminary Modem Networking API. This listing represents a starting point, as additional waveform requirements are defined, the list of interfaces would grow.



3.1.2.3 Theory of Operation:

Using the Super Networking API approach, a single interface for each layer of the network stack would be established. This interface would have to be robust enough to support all existing waveforms and all future waveforms targeted for the JTRS product line.

The Super Networking API is hard, fast and well defined, meaning all interfaces would have defined data typed. The only way to change a Super Networking API would be to update the

interface with each revision of the SCA. New waveforms would be forced to operate within the boundaries of the existing Networking API definitions

With respect to the Link and Network the super Networking API approach may meet the needs of the JTRS user community. The utility of the super NPAI approach with respect to the modem interface is questionable.

The Modem Networking API presents several problems:

- There are a variety of modem products available. The configuration and control interfaces required by the Link layer software to ensure an orderly flow of data are often low-latency interfaces and unique to each hardware device.
- There are a variety of waveforms that need support. The Modem Networking API will most likely be required to support all JTRS waveforms, both networking and non-networking. The Modem Networking API would therefore need to be inclusive of all waveform unique interfaces.
- Each of the interfaces defined in IDL will impact the Core Framework (CF)

It should be noted that the adoption of a Super Networking API approach may invalidate the 2A prototypes currently in development.

3.1.2.4 OE

This approach requires support within the OE to help negotiate interfaces during resource instantiation.

The detailed structure of this approach means that some Networking API mechanisms may be chosen that lie outside the approach used by the OE (i.e. do not use the message interface). The CF may have to be modified as a result.

3.1.2.5 Protocol Layer Applicability

The adoption of a single interface at the Link and or Network layers seems like a reasonable expectation. There are commercially available API definitions, such as DLPI and NPI, that may satisfy this need. However at the Modem (physical) layer a single unified interface definition may be difficult to achieve.

3.1.2.6 Networking API FOM Analysis

3.1.2.6.1 Waveform Portability

Does the Networking API functional and representation definition approach foster waveform portability?

Yes, by using, at each layer, a single Networking API for all waveform applications, impact to the CF is minimized and therefore compatibility problems between various platforms should also be minimized.

Note: It is assumed that 'waveform portability' means porting a complete waveform (and no hardware) from one JTRS platform to another, and not mixing and matching waveform components.

3.1.2.6.1.1 Identify the porting steps required to take a waveform to another JTRS Platform and evaluate the impact that the Networking API functional and representation definition approach has on that process.

3.1.2.6.1.1.1 Porting Classification

Porting activities are identified in the following table.

Stimulus	**Description**	**Classification**
Environment Change:Different Processor	Waveform must be ported to a different processor	Non-recurring
Environment Change:Different OS	Waveform must be moved to a different OS	Non-recurring
Environment Change:Different Programming Language	Waveform Objects must interact with objects on the new platform that are written in a different language	Non-recurring
Environment Change: Different Platform Hardware	Waveform is hosted on new platform hardware that may require additional or different control, e.g. modem, INFSOEC, red I/O	Non-recurring
Environment Change: Different Transport Mechanism	Waveform is hosted on a new platform that uses a different transport mechanism than the source platform	Non-recurring

3.1.2.6.1.1.2 Porting Steps

Each porting activity will require one or more of the following steps. Evaluate each Networking API approach with respect to whether or not that approach causes additional steps to be required in executing the waveform port.

Porting Step	**Description**	**Effort Level**
Direct Rehost	Same processors, OSs etc. – No changes required	None
Recompile	Recompile of those portions of the waveform that are hosted on a different processor	Minimal
Translation Shim	Does the Networking API approach require that shims be added to the waveform to port to the new platform?	Moderate
Edit/ReWrite	Does the Networking API approach require such drastic code changes that edit/rewrite is necessary?	Moderate-Extensive

The following table lists the projected porting steps (from x.6.1.1.2) for each type of stimulus (from x.6.1.1.1):

Stimulus	***Projected Porting Step(s)***
Environment Change: Different Processor (assume same hardware platform, OS, transport mechanism & languages)	***No Networking API effect or recompile depending on the transfer mechanism.***
Environment Change: Different OS (assume same hardware platform, processor, transport mechanism & languages)	***No Networking API effect.***
Environment Change: Different Programming Language (assume same hardware platform, processor, transport mechanism & OS)	***No Networking API effect.***
Environment Change: Different Platform Hardware (assume same hardware platform, processor, OS, transport mechanism & language)	***No Networking API effect.***.
Environment Change: Different Transport Mechanism (assume same hardware platform, processor, OS & language)	***No Networking API effect or recompile depending on the transfer mechanism. Possibility that it could be Direct Rehost*** *if representation does not change* ***and slim possibility that it could be Edit/ReWrite*** *if buffer management, latencies, operational semantics (e.g. half duplex versus full duplex) change.*

3.1.2.6.1.2 Does the Networking API Service Definition foster waveform portability for software that already supports an existing interface, e.g. ARINC 732?

No, a Super Networking API approach is unlikely to be plug-and-play compatible with any existing standard.

3.1.2.6.1.3 What is the impact of the Networking API Service Definition approach on the current SCA definition? And does it affect portability?

3.1.2.6.1.3.1 What is the impact of the Networking API functional and representation definition approach on the current SCA definition?

1. Does the Networking API approach change the definition of any existing interfaces? *This approach would require the CF to define all Super Networking API operations in the IDL interfaces. Based on this, the discovery of new operations would require update of the IDL.*

2. How many new interfaces are created by the Networking API approach? *Many as they are discovered.*

3. What is the impact of the Networking API approach on the Core Framework? *The core framework would be required to support all the interfaces defined in the Super Networking API.*

3.1.2.6.1.3.2 What is the impact of the Networking API functional and representation definition approach on the current or proposed SCA Configuration Control Board?

Any proposed changes or extensions to the existing Networking API would have to be reviewed and approved by the SCA Configuration Control Board.

Major advantage:

Reduce the number of specific Networking API implementations

Useful in addressing security issues in monitoring the control/status bypass.

Major disadvantage:

The complexity and the penalty that small waveforms have for supporting excess code that is not used.

Any time that a new operation is identified the IDL must be updated. Considered an unmanageable premise.

3.1.2.6.2 Able to support new networking and non-networking waveforms

No. This option would be defined based on the current waveforms. New functionality would have to be incorporated into a revision of the SCA.

3.1.2.6.2.1 Does the Networking API Service Definition approach allow incorporation of extensions for new functionality?

Limited. A Super Networking API approach may allow a developer to "pick and choose" features as required, but new functionality would have to be incorporated into a revision of the SCA.

3.1.2.6.2.2 Does the Networking API Service Definition approach support future waveforms in a consistent manner (e.g. reuse of the Networking API for multiple future waveforms)?

To some extent. Since future waveforms would be required to implement functionality using a pre-defined interface, future waveform would be supported in a consistent manner. However, some procedure would be needed to co-ordinate updates to the Networking API standards.

3.1.2.6.2.3 Does the Networking API Service Definition approach support creation of Networking APIs that can be used by networking and non-networking waveforms?

Yes, to the extent that a single Networking API can support a wide variety of waveforms. Non-networking waveforms would be most concerned with the Modem Networking API and care should be taken to ensure that the interface does not prohibit it's use in a non-networking way.

3.1.2.6.2.4 Is there a projected waveform that cannot be supported by the Networking API Service Definition approach?

This is probably the greatest risk with the Super Networking API approach. It is nearly impossible to say whether or not all waveforms can be implemented using a single interface definition.

3.1.2.6.3 Able to support legacy networking and non-networking waveforms

3.1.2.6.3.1 Does the Networking API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking API?

No, a single Networking API used for all waveforms cannot be optimized for any single application. Having to work "within the bounds" of an existing interface definition will invariably result in "work around" situations.

A detailed Networking API definition could support efficient transfer of data, however implementing a single API across several different hardware environments could produce wildly different performance figures.

Approach allows for optimization for all known (projected) waveforms. The risk with this approach is missing some operation.

3.1.2.6.3.2 Does the Networking API Service Definition approach allow incorporation of extensions for new functionality?

Limited. A Super Networking API approach may allow a developer to "pick and choose" features as required, but new functionality would have to be incorporated into a revision of the SCA.

Ideally, the Super Networking API would not require extensions. As new functionality is identified the SCA would have to be revised to incorporate this new functionality.

Some type of mechanism for control of these extensions, if allowed, would be required. A question would be a what rate would the extensions be allowed. What would be the affect of extensions on fielded units.

3.1.2.6.3.3 Does the Networking API Service Definition support legacy networking waveforms in a consistent manner (e.g. reuse of the Networking API for multiple legacy waveforms)?

Support for legacy waveforms (network and non-network) are possible but the inclusion of all exponentially increases the overall complexity of the Super Networking API. In addition, in some cases short circuit of one or more of the protocol layers may be required causing additional complexity.

3.1.2.6.3.4 Does the Networking API Service Definition support creation of Networking APIs that can be used by networking and non-networking waveforms?

A Super Networking API approach does not prohibit use by networking or non-networking waveforms. But the fact that all waveform applications would be required to use a single interface definition does complicate the Networking API definition.

It becomes a complicated operation because all the operations to support both networking and non-networking waveforms must be included.

3.1.2.6.3.5 Is there a known waveform that cannot be supported by the Service Definition approach?

No.

3.1.2.6.4 Performance (Efficiency/Latency)

3.1.2.6.4.1 Does the Networking API functional and representation definition approach support low-overhead, efficient transfer of control and traffic across the Networking API?

A detailed Networking API definition could support efficient transfer of data, however implementing a single API across several different hardware environments could produce wildly different performance figures.

-- also --

See answer to x.6.3.1.

3.1.2.6.4.2 Does the Networking API functional and representation definition approach support high data throughput, low latency control?

The Super Networking API would have to have provisions for handling high data rates. There is nothing in the super Networking API approach that would prohibit the creation of high data rate interface definitions.

Passed by reference would be part of what would be supported by this alternate.

3.1.2.6.4.3 How is performance affected by any required message parsing within the producer and consumer resources?

Because everything is explicitly defined (no any's no name/value pairs) there would be no required parsing and thus no impact.

3.1.2.6.5 Extensibility

3.1.2.6.5.1 Does the Networking API functional and representation definition approach allow Networking API definition to be extended to support new waveform interfaces, functionality, representation, and mechanisms?

No. The purpose of a Super Networking API is to eliminate the need for waveform-unique interface extensions.

Yes, see answer to x.6.2.1.

3.1.2.6.5.2 Does a mechanism(s) exist to limit proliferation of published Networking API extensions?

Yes, the adoption of the Super Networking API approach will eliminate (limit) the proliferation of Networking API extensions. But the evolution of the SCA may require should method for extensions and backward compatibility.

3.1.2.6.6 Networking API Reuse at same layer

3.1.2.6.6.1 Does the Networking API functional and representation definition approach foster Networking API reuse at the same layer, e.g. use the same modem Networking API definition for multiple (different) waveforms?

Yes, a Super Networking API approach would require Networking API reuse since all waveform would be required to adopt the same Networking API definitions. An issue with the Super Networking API is what is done with unsupported operations. If all operations must be supported, a large amount of software overhead is envisioned that is required but may never be used.

3.1.2.6.6.2 Does the Networking API functional and representation definition approach foster commonality of interface?

1. Common Operation Set? *Yes. Because of the single set of operations with the same parameters.*

2. Common Parameter Set? *Yes. The Super Networking API has a single parameter set.*

3. Common Message/Data Format? *Yes. The Super Networking API, by definition has a common message/data format.*

3.1.2.6.7 Component Interoperability (within the box, component plug and play)

3.1.2.6.7.1 Does the Networking API functional and representation definition approach foster development of Networking APIs that allow individual waveform components to be reused by other waveforms?

Yes, all networking components would adhere to a single set of API definitions. This is one of the strengths of the super Networking API approach.

An issue is the consistent implementation (another layer of standardization) for ensuring what is required to be implemented and what could be left out for a particular waveform.

3.1.2.6.7.2 Do the Networking API definition, implementation, and configuration control methods support component interoperability?

1. Does the Networking API approach support interface negotiation (e.g. Global User ID)? *Yes. There will be a need for tracking which subset of the Super Networking API commands are in use*.

2. Does the Networking API approach support interface version query? *Yes. As part of the definition of the Super Networking API, the approach would support this operation.*

3. Does the Networking API approach support backward interface compatibility? *Yes. By definition, the Super Networking API includes all of the legacy and future operations and parameters.*

3.1.2.6.8 Cost

3.1.2.6.8.1 Is the Networking API Service Definition a cost driver??

1. Implementation Cost? *Implementation cost is relatively high, because a single Networking API would be more complex than a waveform specific one and thus would require a greater number of hours than a specific waveform Networking API for implementation.*

2. Maintenance Cost? *Moderate, a super Networking API approach would generate a continuously evolving standard. Waveform applications may need to "upgrade" to newer standards to remain compatible.*

3. Porting Cost? *Low, since there would be a single Networking API standard at each layer, application porting issues should be well documented and understood for every JTRS compliant product.*

It different applications don't implement all of the operations, then the cost may be high.

Two ways to lay out the requirements: 1) specify that all operations must be supported either implementing the function or an exception. All applications know that all operations will get a response. 2) waveform implements only certain operations and then it is not known what operations are supported on a port.

3.1.2.6.9 Commercial Acceptance

3.1.2.6.9.1 Does the Networking API functional and representation definition approach have current or projected commercial acceptance, (e.g. SDRF, IEEE, OMG, or defacto)?

1. No commercial acceptance?

2. Some commercial acceptance?

3. Widespread commercial acceptance?

The Super Networking API approach would have no commercial acceptance. At the higher OSI levels (Link and Network) it might be possible to implement a commercial standard for many waveforms (DLPI, for Link for example). However, since the Super Networking API is required to support all JTRS waveforms, there would most likely be no suitable commercial standard. Since the super Networking API would be larger and more complex than what is required for

any particular waveform, it is unlikely that a commercial venture would adopt it because of the excess code that is included with the Super Networking API. For waveforms that require a very small subset of the Super Networking API there would be a very large overhead penalty to carry along all of the excess code. This would be driver for memory requirements on this small subsets.

3.1.2.6.9.2 Does the Networking API functional and representation definition approach foster the adoption of Networking APIs with current or projected commercial acceptance, (e.g. DLPI, Sockets, defacto (Microsoft))?

1. No commercial acceptance?

2. Some commercial acceptance?

3. Widespread commercial acceptance?

No, it is unlikely that any existing interface definition would be suitable for all JTRS waveform.

3.1.2.6.10 Security

3.1.2.6.10.1 Do the JTRS security requirements prevent the use of this Networking API definition or implementation?

1. Use of IDL Networking API definition if red and black objects can only communicate through INFOSEC adapters? *This is a question if all messages flowing from red to black must pass through the generic message interface of an INFOSEC adapter. How can IDL interface definitions be used between red and black objects?*

No, a single Super Networking API would simplify security issues. It would be a big set but it would known and quantifiable.

3.1.2.6.11 Ability to Adapt to Commercial Trends

3.1.2.6.11.1 Is the Networking API functional and representation definition approach able to adapt to emerging commercial networking standards?

No, once an interface definition is established, newer versions could not adopt new emerging networking standards. That would invalidate existing waveform products.

3.1.3 Option 3 - Commercial Model Networking API Service Definition

Interface Type: Networking API Service Definition

Name: Option 3 – Commercial Model Networking API Service Definition

Date: 20 March 2000

Revision: 3.0

Authors: Jim Stevens (jasteven@collins.rockwell.com) &
Jackson Anderson (ajanders@collins.rockwell.com)

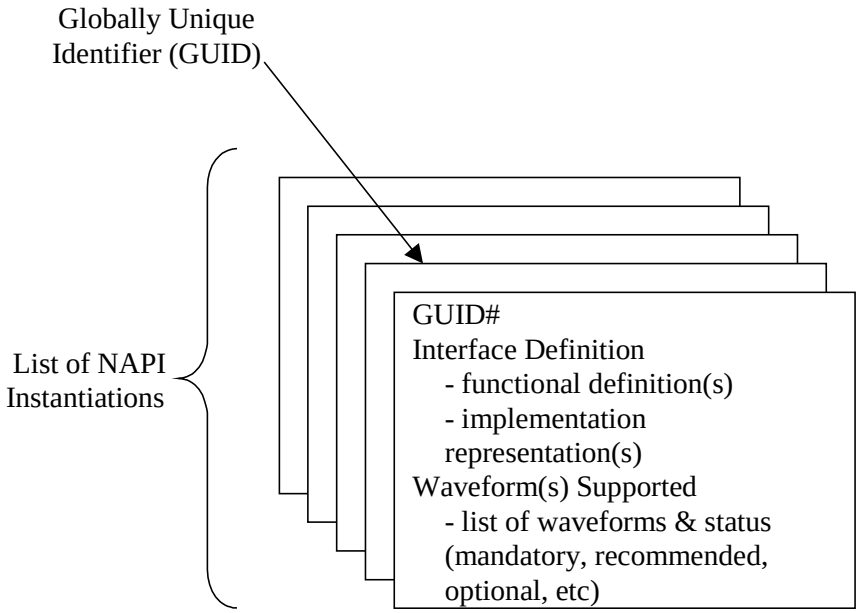
Company: Rockwell Collins

3.1.3.1 Description

This approach supports the flexibility of the way the commercial world creates APIs combined with a configuration control process (similar to that used for controlling MIBs in the SNMP protocol) to support interoperability and portability. Similarly to how commercial networking vendors can create their own APIs or support standard or de facto APIs, JTRS waveform implementers can create their own Networking APIs or use pre-defined Networking APIs. If a waveform implementer creates a new Networking API, they register it with a JTRS Networking API configuration control board. The JTRS Networking API configuration board determines the status (i.e. Mandatory, Recommended, Other). This Networking-API Networking API Service Definition was presented at JTRS Step 2A IPR#3.

3.1.3.2 Diagram

The following diagram shows the generic and flexible concept behind this Networking API.



3.1.3.3 Theory of Operation

This Networking-API Networking API Service Definition alternative defines a method of interface configuration control, interface version identification, and interface negotiation. It was designed to give vendors and users great freedom and flexibility in trading off performance, portability and cost drivers. It promotes open interface standards for waveforms and allows customers to assess the impact of changes. Its configuration control provides mechanism for managing evolution and tracking changes. The flexible service definition was modeled after the Microsoft COM/DCOM approach to service definitions and the configuration control approach was modeled after that used to manage MIBs for SNMP and CMIP.

A Networking-API Service Definition will be defined at the functional level for a waveform or set of waveforms. The developer of a waveform implementation will publish the Networking APIs of its visible network components. Each Networking-API Service Definition will be assigned a globally unique identifier (GUID) that:

- Specifies the defined Networking-API service interface
- Specifies the version number of the interface
- Must be visible to the Domain Manager profile.

There will be a controlling body for GUIDs that controls the control assignment of GUIDs. This configuration control body should be an open standards body similar to the IETF or the IEEE. This controlling body would assign a status for each Networking-API Service Definition similar to the way that the IETF assigns statuses to Internet standards today. Potential statuses could include:

- mandatory for a particular waveform or set of waveforms (i.e. if a vendor implements one of the particular waveforms, then they must implement this Networking-API Service Definition to be JTRS compliant)
- recommended for a particular waveform or set of waveforms (i.e. if a vendor implements one of the particular waveforms, then it is recommended, but not required, that they implement this Networking-API Service Definition to be JTRS compliant)
- allowable for a particular waveform or set of waveforms (i.e. the vendor is allowed, but not required to implement this Networking-API Service Definition to be JTRS for one of the particular waveforms)
- Other possible statuses including experimental, not recommended, etc.

Any Networking-API Service Definitions that are chosen as JTRS standards (i.e., mandatory or recommended) must be published and non-proprietary. However, vendors can choose to keep the details of particular Networking APIs private or proprietary (so that the status is likely to be allowable or some other.) The owner of interface controls version numbering. This is similar to the approach used to managing MIBs in SNMP and CMIP.

For example, suppose that there is a JTRS mandatory Networking-API Service Definition for the SINCGARS waveform. If a vendor develops a new proprietary way to mitigate co-site interference that requires a new SINCGARS Networking-API Service Definition, then that vendor could get a GUID for its new Networking-API Service Definition. The vendor would have the freedom to keep the details of the new Networking-API Service Definition proprietary or to make it open. However, the new Networking-API Service Definition could only become

mandatory or recommended if the vendor made the Networking-API Service Definition open. In either case, the vendor could now develop and sell a SINCGARS modem object that supports both the mandatory Networking-API Service Definition and this new allowable Networking-API Service Definition. Since there is a GUID, the Domain Manager can determine at instantiation whether the SINCGARS modem object is being linked together with other objects that know how to use the allowable SINCGARS service definition. If other such objects were available, then a better co-site mitigation JTRS radio would result. If the other objects only knew how to support the mandatory Networking-API Service Definition, then an operational JTRS radio would result that runs the SINCGARS waveform.

All visible interfaces will be uniquely identified. A mechanism will be provided within the Domain/Resource Manager to identify the interfaces used by the resources. When the Domain/Resource Manager instantiates a Resource to satisfy a User request it selects the resource with matching interface.

3.1.3.4 Operational Environment (OE)

This approach requires support within the OE resource profiles for GUIDs so that it can determine which resources have compatible Networking-API Service Definitions when instantiating waveforms.

Waveform developers are free to define their Networking-API Service Definitions using the OE Message Interface or to develop a new CORBA interface that is not part of the OE. In either case, since it is defined as part of the Networking-API Service Definition, two objects that implement the Networking-API service defined by a given GUID will be interoperable.

3.1.3.5 Protocol Layer Applicability

This Networking-API Service Definition alternative is applicable to any protocol layer due to great flexibility.

3.1.3.6 Networking-API FOM Analysis

3.1.3.6.1 Waveform Portability

Does the Networking-API Service Definition approach foster waveform portability?

Yes, if two implementations use the same Networking-API definition. They are most likely to do so for Networking-API definitions that are made "mandatory" or "recommended."

Note: It is assumed that 'waveform portability' means porting a complete waveform (and no hardware) from one JTRS platform to another, and not mixing and matching waveform components.

3.1.3.6.1.1 Identify the porting steps required to take a waveform to another JTRS Platform and evaluate the impact that the Networking-API Service Definition approach has on that process.

The following table lists the projected porting steps for a wide range of porting stimulus.

Porting Stimulus	**Description**	***Projected Porting Steps Required Because of the Networking-API Networking API Service Definition***
Environment Change: Different Processor	Waveform must be ported to a different processor	No Networking-API effect.
Environment Change: Different OS	Waveform must be moved to a different OS	No Networking-API effect.
Environment Change: Different Programming Language	Waveform Objects must interact with objects on the new platform that are written in a different language	No Networking-API effect.
Environment Change: Different Platform Hardware	Waveform is hosted on new platform hardware that may require additional or different control, e.g. modem, INFSOEC, red I/O	No Networking-API effect.
Environment Change: Different Transport Mechanism	Waveform is hosted on a new platform that uses a different transport mechanism than the source platform	Direct Rehost **or** Translation Shim **depending on the transport mechanism.** (Note: the IDL/ORB abstracts interfaces to other objects.) If the transport is not abstracted by the ORD it will require a Translation Shim or Edit/Rewrite

The projected porting steps are defined from least to most effort in the following table.

Porting Step	**Description**	**Effort Level**
Direct Rehost	Same processors, OSs etc. – No changes required	None
Recompile	Recompile of those portions of the waveform that are hosted on a different processor	Minimal
Translation Shim	Does the Networking-API approach require that shims be added to the waveform to port to the new platform?	Moderate
Edit/ReWrite	Does the Networking-API approach require such drastic code changes that edit/rewrite is necessary?	Moderate-Extensive

3.1.3.6.1.2 Does the Networking-API Service Definition approach foster waveform portability for software that already supports an existing interface, e.g. ARINC 732?

No, this approach does not foster portability between Networking-API Service Definitions for the same waveform because it does not force the waveform developer to use the same operation(s), parameters, signals, etc. Since this approach does not impose a minimal set of operations (such as the Small Service Definition Using Name/Value Pairs), it will be more likely to require an Edit/ReWrite than it is to require a Translation Shim.

3.1.3.6.1.3 What is the impact of the Networking-API Service Definition approach on the current SCA definition? And does it affect portability?

1. Does the Networking-API approach change the definition of any existing interfaces? *No.*

2. How many new interfaces are created by the Networking-API approach? *The number of new interfaces can vary for each different Networking-API Service Definition. Any new interfaces will be defined outside the Core Framework and thus "bypass the core framework." Since this Networking-API approach can use the Core Framework interfaces "as is" or defined new interfaces "outside" the Core Framework, this Networking-API approach does not impact the Core Framework*

3. What is the impact of the Networking-API approach on the Core Framework? *The Core Framework will have to be able to track Networking-API Service Definition GUIDs and version numbers in resource profiles so that it can instantiate resources that can interoperate.*

3.1.3.6.2 Able to support new networking and non-networking waveforms

Yes, by modifying already developed Networking-API Service Definitions and changing the version number of creating a new Networking-API Service Definition and getting a new GUID.

3.1.3.6.2.1 Does the Networking-API Service Definition approach allow incorporation of extensions for new functionality?

Yes, by creating a new Networking-API Service Definition version or Networking-API Service Definition with a new GUID.

3.1.3.6.2.2 Does the Networking-API Service Definition approach support future waveforms in a consistent manner (e.g. reuse of the Networking-API for multiple future waveforms)?

Not really, since this Networking-API Service Definition approach has complete freedom and flexibility to define whatever an implementer needs. However, the configuration board can impose consistency by limiting what it really depends upon the particular service chosen for a given Networking-API Service Definition.

3.1.3.6.2.3 Does the Networking-API Service Definition approach support creation of Networking-APIs that can be used by networking and non-networking waveforms?

Yes, since this Networking-API Service Definition approach has complete freedom and flexibility to define whatever an implementer needs.

3.1.3.6.2.4 Is there a projected waveform that cannot be supported by the Networking-API Service Definition approach?

No, since this Networking-API Service Definition approach has complete freedom and flexibility to define whatever an implementer needs.

3.1.3.6.3 Able to support legacy networking and non-networking waveforms

Yes, through its complete freedom and flexibility to define whatever an implementer needs.

3.1.3.6.3.1 Does the Networking-API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking-API?

Yes, through its complete freedom and flexibility to define whatever an implementer needs.

3.1.3.6.3.2 Does the Networking-API Service Definition approach allow incorporation of extensions for new functionality?

Yes, through its complete freedom and flexibility to define a new Networking-API Service Definition for extensions. The new Networking-API Service Definition could be a simple change of an existing Networking-API Service Definition (in which case the version number is likely to be incremented rather than assigning a new GUID). Alternatively, the new Networking-API Service Definition could be sufficiently different from other Networking-API Service Definitions that a new GUID is assigned. Note that inheritance is allowed, but not required for this approach.

3.1.3.6.3.3 Does the Networking-API Service Definition approach support legacy networking waveforms in a consistent manner (e.g. reuse of the Networking-API for multiple legacy waveforms)?

Not really since the Networking-API will let legacy waveforms continue to use their existing interface by assigning a GUID to that given interface and publishing the interface if it is mandatory or recommended. However, the configuration control board can pursue such consistency by limiting what it assigns the mandatory or recommended statuses to.

3.1.3.6.3.4 Does the Networking-API Service Definition approach support creation of Networking-APIs that can be used by networking and non-networking waveforms?

Yes, through its complete freedom and flexibility to define whatever an implementer needs.

3.1.3.6.3.5 Is there a known waveform that cannot be supported by the Networking-API Service Definition approach?

No, since the Networking-API has complete freedom and flexibility to define whatever an implementer needs.

3.1.3.6.4 Performance (Efficiency/Latency)

3.1.3.6.4.1 Does the Networking-API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking-API?

Yes, through its complete freedom and flexibility to define whatever an implementer needs.

3.1.3.6.4.2 Does the Networking-API Service Definition approach support high data throughput, low latency control?

Yes, through its complete freedom and flexibility to define whatever an implementer needs.

3.1.3.6.4.3 How is performance affected by any required message parsing within the producer and consumer resources?

It will depend upon the actual representation chosen for a particular Networking-API definition. If the approach is IDL specific then it would at least as fast as any other approach. The implementer is given the freedom to develop an operations based Networking-API Service Definition rather than a name/value based definition which would minimize any required message parsing.

3.1.3.6.5 Extensibility

3.1.3.6.5.1 Does the Networking-API Service Definition approach allow Networking-API definition to be extended to support new waveform interfaces, functionality, representation, and mechanisms?

Yes, through its complete freedom and flexibility to define a new Networking-API Service Definition for extensions. The new Networking-API Service Definition could be a simple change of an existing Networking-API Service Definition (in which case the version number is likely to be incremented rather than assigning a new GUID). Alternatively, the new Networking-API Service Definition could be sufficiently different from other Networking-API Service Definitions that a new GUID is assigned. Note that inheritance is allowed, but not required for this approach.

3.1.3.6.5.2 Does a mechanism(s) exist to limit proliferation of published Networking-API extensions?

No, this Networking-API approach per se does not have any mechanisms to limit proliferation of published Networking-API extensions. However, this approach assumes that there is a standard configuration control body that determines which published Networking-API extensions are "mandatory" and "recommended" for a given waveform. In that sense it is very similar to SNMP which will allow a plethora of MIB definitions, but has the IETF define which MIBs are the standard MIBs that all routers/hosts must/should support.

3.1.3.6.6 Networking-API Reuse at same layer

3.1.3.6.6.1 Does the Networking-API Service Definition approach foster Networking-API reuse at the same layer, e.g. use the same modem Networking-API definition for multiple (different) waveforms?

This approach does not prohibit but does not foster reuse since it does not mandate common name/value pairs. The implementer is allowed, but not required to provide such reuse. Also, the configuration control body could foster such reuse through what Networking-API Service Definition it chooses to label as mandatory, recommended, or other.

3.1.3.6.6.2 Does the Networking-API Service Definition approach foster commonality of interface?

1. Common Operation Set? *No (although the standard configuration control body could foster such commonality).*

2. Common Parameter Set? *No (although the standard configuration control body could foster such commonality).*

3. Common Message/Data Format? *No (although the standard configuration control body could foster such commonality).*

3.1.3.6.7 Component Interoperability (within the box, component plug and play)

3.1.3.6.7.1 Does the Networking-API Service Definition approach foster development of Networking-APIs that allow individual waveform components to be reused by other waveforms?

No, the Networking-API itself does not per se, but the concept of a standard configuration body coupled with the globally unique IDs does foster such development.

3.1.3.6.7.2 Do the Networking-API definition, implementation, and configuration control methods support component interoperability?

1. Does the Networking-API approach support interface negotiation (e.g. Global User ID)? *Yes*.

2. Does the Networking-API approach support interface version query? *Yes.*

3. Does the Networking-API approach support backward interface compatibility? *Yes, through GUID & version identification. In order to support backward compatibility, the waveform needs to support previous versions of the same Networking-API Service Definition or multiple different Networking-API Service Definitions for the same waveform.*

3.1.3.6.8 Cost

3.1.3.6.8.1 Is the Networking-API Service Definition approach a cost driver?

1. Implementation Cost? *The same or lower than "pure IDL" since the implementer has complete freedom to define the interface within the constraints of giving it a GUID.*

2. Maintenance Cost? *Higher than "pure IDL" since the implementer will constantly have to learn new Networking-API service approaches.*

3. Porting Cost? *Likely to be higher than "pure IDL" since there will be a good chance that another resource may have implemented a different Networking-API. (Note however, that the use of "mandatory" and "recommended" Networking-APIs could lower the porting cost down to slightly higher than pure IDL.)*

3.1.3.6.9 Commercial Acceptance

3.1.3.6.9.1 Does the Networking-API Service Definition approach have current or projected commercial acceptance, (e.g. SDRF, IEEE, OMG, or de facto)? (No, some, or widespread commercial acceptance?)

Many wireless standards are being developed due to the rapid explosion in commercial wireless products. Although we do not know which of these commercial initiatives will achieve commercial de facto standard acceptance similar to the Hayes modem AT command set, several such interface de facto standards are bound to arise. The one that wins is probably going to be based around getting initial market penetration in both the commercial cellular data market, the wireless LAN market, and the consumer home/personal network market. Since this Networking-API is being defined for JTRS which is focused on military applications, it is likely that this Networking-API (nor any Networking-APIs) will achieve widespread commercial acceptance. Although it is not likely that any JTRS Networking-API will achieve some commercial acceptance (assuming there are no technical problems with the Networking-API). Since the JTRS addresses a large, important military market (although small in comparison to the commercial market), any JTRS Networking-API that does not have technical problems is likely to be accepted by the SDRF and/or the IEEE.

Since, this Networking-API approach allows the reuse of existing commercial interfaces for particular waveforms, it can get some commercial acceptance just by tracking with existing commercial acceptances. Note that the high flexibility of this approach is likely to be a major issue for a significant minority of engineers.

Thus, the combination of the exploding commercial wireless market coupled with the fact that this Networking-API approach has not technical negatives means that this approach, if selected as the JTRS Networking-API, would likely achieve some commercial acceptance and become a SDRF and/or IEEE standard.

3.1.3.6.9.2 Does the Networking-API Service Definition approach foster the adoption of Networking-APIs with current or projected commercial acceptance, (e.g. DLPI, Sockets, or de facto (Microsoft))? (No commercial acceptance? Some commercial acceptance? Or widespread commercial acceptance?)

Yes, this approach allows the Networking-API to adopt current and projected Networking-APIs that are already in use.

3.1.3.6.10 Security

This section requires further work and coordination with the Security IPT.

3.1.3.6.10.1 Do the JTRS security requirements prevent the use of this Networking-API definition or implementation?

1. Use of IDL Networking-API definition if red and black objects can only communicate through INFOSEC adapters? *This is a question if all messages flowing from red to black must pass through the generic message interface of an INFOSEC adapter. How can IDL interface definitions be used between red and black objects?*

2.

A complete answer will await meeting with the Security IPT, but there is likely to be a problem with approach in passing control from Red to Black due to the unbounded nature of the interface with respect to functional definitions and representation.

3.1.3.6.11 Ability to Adapt to Commercial Trends

3.1.3.6.11.1 Is the Networking-API Service Definition approach able to adapt to emerging commercial networking standards?

Yes, due to its high flexibility.

3.1.4 Option 4 - Commercial Model with Inheritance Networking API Service Definition

Interface Type: Networking API Service Definition

Name: Option 4 – Commercial Model with Inheritance Networking API Service Definition

Date: 31 May 2000

Revision: 9.0

Author: Fred Mabe (fdmabe@collins.rockwell.com)

Company: Rockwell Collins

3.1.4.1 Description

The Networking API Service Definition for a waveform protocol layer details the operations (primitives), the attributes (variables), their representation (structures, types, formats) and its behavior. In order to obtain interoperability and interchangeability of the network protocol entities standard and consistent interfaces are required. From a programmer's perspective, to reuse/port a existing network protocol entity, the Networking API Service Definition is the most important part of the Networking API. It is important that the service definition provide inter-operability independent of the actual software implementation of those entities. Networking API Service Definitions are a method to document these interfaces. Creation of Networking API Service Definitions via inheritance, supports the flexible object oriented approach used by the commercial world. An object oriented approach facilitates polymorphism and code reuse. Like C++ and other object oriented languages there must be a method to foster inheritance and diminish the propensity of growth of inheritance trees. Figure 1 represents the path to reach common Networking API Service Definitions via inheritance.

3.1.4.2 Diagram

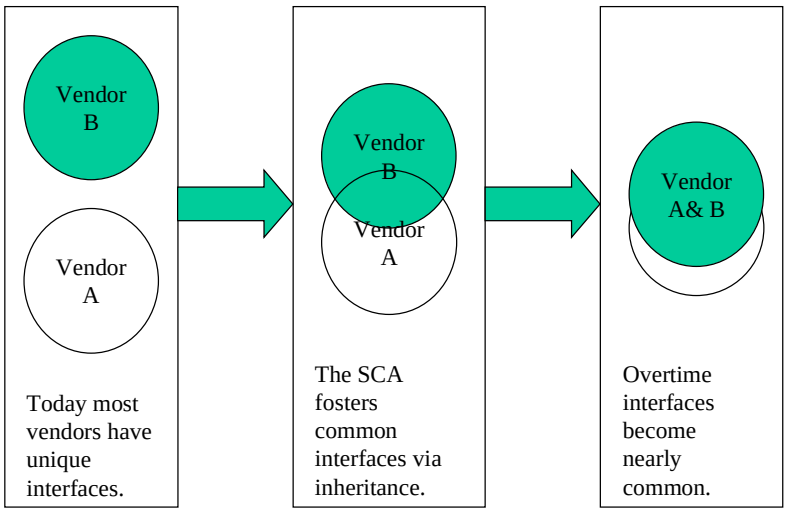


Figure -1 Interface Convergence Diagram

3.1.4.3 Theory of Operation

The Interface Definition Language (IDL) will be used to define Networking API Service Definitions to foster inheritance and interoperability. IDL provides a method to inherit from multiple interfaces. IDL also separates implementation from specification. In addition, the behavior and the states must be defined in the Networking API Service Definitions, to augment the IDL, so the network protocol implementation can be abstracted as a black box. A Networking API Service Definition is a contract between the service user and the service provider (e.g. Link layer and a Modem). A particular Networking API Service Definition is uniquely identified by a Universally Unique Identifier (UUID). Similarly to how commercial networking vendors can create their own APIs or support standard or de facto APIs, waveform implementers can create their own Networking API Service Definitions or use previously defined Networking API Service Definitions. If a waveform implementer creates a new Networking API Service Definition, they publish it and its associated UUID. Alternatively if an implementor creates private Networking API Service Definition he may publish his UUID but not the service definition. It is recommended that public Networking API Service Definitions be controlled by a configuration board to control the propensity of growth of inheritance trees. The UUID used by the Domain Manager to insure interoperability of serviced definitions shall be a single UUID that identifies complete single interface that is the combination of inherited interface packages from

the inheritance tree rather than the individual interface packages in the inheritance tree. For example, two service definitions may be inherited to form a new Networking API serviced which is a superset of the two inherited Networking API Service Definitions. This superset Networking API Service Definition must be given a unique UUID separate from the inherited Networking API Service Definitions. Creation of a Networking API Service Definition shall be accomplished via one the following methods:

1. Reuse an existing Networking API Service Definition: If an existing Networking API Service Definition is identical to the new protocol entity's' interface, or the new protocol entity can easily be mapped to an existing Networking API Service Definition, the Networking API Service Definition shall be reused. How these Networking APIs are written determines the extent which they can be reused. For example if you define a F/A-18 Networking API Service Definition it will not be very reusable. However if you define F/A-18 which is a collection of multiple interfaces which can be inherited (e.g. situation display, tactical sensors, Navigation, throttle ...) then reuse can occur

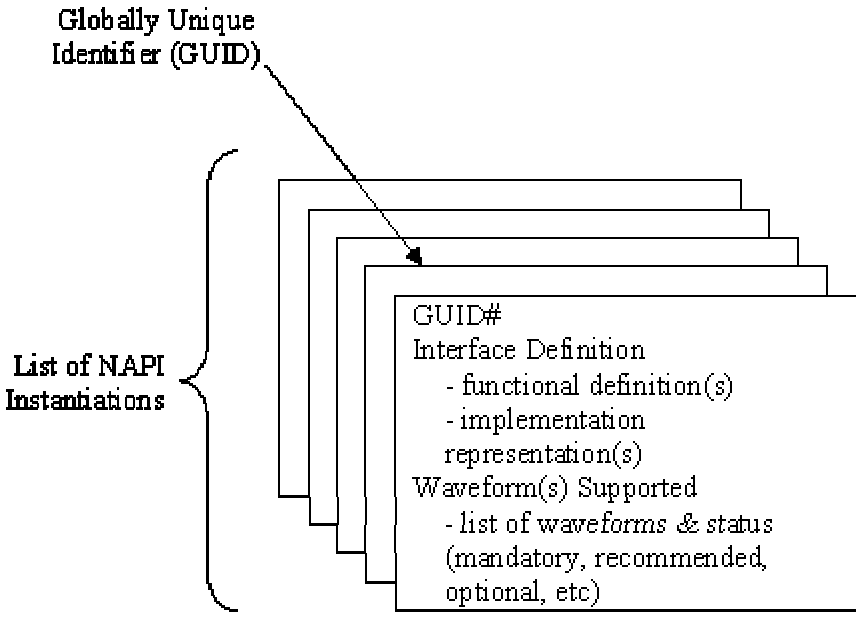
2. Reuse of existing Networking API Service Definitions via inheritance to create an new Networking API Service Definition: The new Networking API interface shall be created from parts of existing interfaces that may be inherited. How these Networking APIs are written determines the extent which they can be reused. For example if you define a F/A-18 Networking API Service Definition it will not be very reusable. However if you define F/A-18 which is a collection of multiple interfaces which can be inherited (e.g. situation display, tactical sensors, Navigation, throttle ...) then reuse can occur

3. Write the existing interface of the network protocol entity in IDL and create a new Networking API Service Definition: If the protocol entity being ported has a standard commercial interface that has not been document via an Networking API Service Definition, import and publish the commercial standard via the Networking API Service Definition process. This will allow commercial COTS software to be ported without modification.

4. Use an IDL wrapper/adapter to hide the incorporation of the new protocol: If porting a network protocol entity which has a tight coupling with another piece of software or hardware, the interface may be hidden via an IDL wrapper. For example if porting an IP and Ethernet software package which is implemented in hardware (e.g. ASIC), a IDL wrapper would be written to the highest layer the ASIC would expose (e.g. network interface). The ASIC would have only a network Networking API Service Definition. Hidden via the IDL wrapper would be the interfaces to the IP protocol and Ethernet. Also if porting an existing network protocol entity, the network protocol entity has two or more layers which are always together and would never be used separately, the interface between these layers may be hidden via an IDL wrapper. However care must be taken in the decision to use an IDL wrapper because it effects the portability of the network protocol entities hidden by the IDL wrapper.

In the above specified methods it is recommended that the implied numerical order of preference be followed. All Networking API Service Definitions shall be written in IDL. The Networking API Service Definitions shall describe behavior and state changes of the network protocol caused by invoking primitives. Parameters are the data associated with the primitives. Networking API Service Definitions may be written in "specific primitives/parameters": when a primitive is invoked the client does not require further interpretation of the parameters to determine the expected behavior (e.g. set_Frequency(in double frequency)), or "name/value pairs": when a primitive is invoked the client does require further interpretation of the parameters to determine

the expected behavior (e.g. command(in Any any_command)). It is recommend to avoid "name/value pairs" and use the "specific primitives/parameters" because of extra processing overhead associated with CORBA marshalling and interpretation of the "name/value pairs". However there are cases when "name/value pairs are advantageous.



Configuration Control of Networking API Service Definition

Configuration control provides mechanism for managing evolution and tracking changes. A Networking API Service Definition will be defined at the functional level for a waveform or set of waveforms. The developer of a waveform implementation will publish the Networking APIs of its visible network components. Each Networking API Service Definition will be assigned a UUID via the a mathematical formula that produces a unique number for each Networking API service interface.

The control/publishing of the Networking API Service Definition could be accomplished in many ways. The following are three example methods:

- Follow Microsoft COM/DCOM's approach that does not require a configuration control body. Basically this approach is survival of the fittest interface. Anyone can publish an interface and overtime the bad ones will not be used.
- Keep the Networking API Service Definitions as part of the SCA and control it via a configuration control body.
- Use a configuration control body where Networking API Service Definitions are publish separate from the SCA; maybe a industry forum like the SDRF

The latter is better method foster inheritance, gain commercial acceptance and control the propensity of growth. There should be a controlling body for *UUID*s that controls the controls publishing of *UUID*s. This configuration control body should be an open standards body similar to the SDRF, IETF, or the IEEE. This controlling body would assign a status for each Networking API Service Definition similar to the way that the IETF assigns statuses to Internet standards today. Potential statuses could include:

- mandatory for a particular waveform or set of waveforms (i.e. if a vendor implements one of the particular waveforms, then they must implement this Networking API Service Definition to be JTRS compliant)
- recommended for a particular waveform or set of waveforms (i.e. if a vendor implements one of the particular waveforms, then it is recommended, but not required, that they implement this Networking API Service Definition to be JTRS compliant)
- allowable for a particular waveform or set of waveforms (i.e. the vendor is allowed, but not required to implement this Networking API Service Definition to be JTRS for one of the particular waveforms)
- Other possible statuses including experimental, not recommended, etc.

Any Networking API Service Definitions that are chosen as JTRS standards (i.e., mandatory or recommended) must be published and non-proprietary. However, vendors can choose to keep the details of particular Networking APIs private or proprietary (so that the status is likely to be allowable or some other.) The owner of interface controls version numbering. This is similar to the approach used to managing MIBs in SNMP and CMIP.

For example, suppose that there is a JTRS mandatory Networking API Service Definition for the SINCGARS waveform. If a vendor develops a new proprietary way to mitigate co-site interference that requires a new SINCGARS Networking API Service Definition, then that vendor could get a *UUID* for its new Networking API Service Definition. The vendor would have the freedom to keep the details of the new Networking API Service Definition proprietary or to make it open. However, the new Networking API Service Definition could only become mandatory or recommended if the vendor made the Networking API Service Definition open. In either case, the vendor could now develop and sell a SINCGARS modem object that supports both the mandatory Networking API Service Definition and this new allowable Networking API Service Definition. Since there is a UUID, the Domain Manager can determine at instantiation whether the SINCGARS modem object is being linked together with other objects that know how to use the allowable SINCGARS service definition. If other such objects were available, then a better co-site mitigation JTRS radio would result. If the other objects only knew how to support the mandatory Networking API Service Definition, then an operational JTRS radio would result that runs the SINCGARS waveform.

3.1.4.4 Operational Environment (OE)

This approach requires support within the OE resource profiles for UUIDs so that it can determine which resources have compatible Networking API Service Definitions when instantiating waveforms.

Waveform developers are free to define their Networking API Service Definitions using the OE Message Interface or to develop a new CORBA interface that is not part of the OE. In either case, since it is defined as part of the Networking API Service Definition, two objects that implement the Networking API service defined by a given UUID will be interoperable.

3.1.4.5 Protocol Layer Applicability

This Networking API Service Definition alternative is applicable to any protocol layer due to great flexibility.

3.1.4.6 Networking API FOM Analysis

3.1.4.6.1 Waveform Portability

Does the Networking API Service Definition approach foster waveform portability?

Yes, by developing a small number of primitives and a framework based upon IDL for encoding the service definition, the Networking API Service Definition can be moved to different platforms.

Note: It is assumed that 'waveform portability' means porting a complete waveform (and no hardware) from one JTRS platform to another, and not mixing and matching waveform components.

3.1.4.6.1.1 Identify the porting steps required to take a waveform to another JTRS Platform and evaluate the impact that the Networking API Service Definition approach has on that process.

3.1.4.6.1.1.1 Porting Classification

The following table lists the projected porting steps for a wide range of porting stimulus.

Porting Stimulus	**Description**	***Projected Porting Steps Required Because of the Networking API Service Definition***
Environment Change: Different Processor	Waveform must be ported to a different processor	*No Networking API effect.*
Environment Change: Different OS	Waveform must be moved to a different OS	*No Networking API effect.*
Environment Change: Different Programming Language	Waveform Objects must interact with objects on the new platform that are written in a different language	*No Networking API effect.*
Environment Change: Different Platform Hardware	Waveform is hosted on new platform hardware that may require additional or different control, e.g. modem, INFSOEC, red I/O	*No Networking API effect.*
Environment Change: Different Transport Mechanism	Waveform is hosted on a new platform that uses a different transport mechanism than the source platform	*Direct Rehost* ***or*** *Translation Shim* ***depending on the transport mechanism.***

The projected porting steps are defined from least to most effort in the following table.

Porting Step	**Description**	**Effort Level**
Direct Rehost	Same processors, OSs etc. – No changes required	None
Recompile	Recompile of those portions of the waveform that are hosted on a different processor	Minimal
Translation Shim	Does the Networking API approach require that shims be added to the waveform to port to the new platform?	Moderate
Edit/ReWrite	Does the Networking API approach require such drastic code changes that edit/rewrite is necessary?	Moderate-Extensive

3.1.4.6.1.2 Does the Networking API Service Definition foster waveform portability for software that already supports an existing interface, e.g. ARINC 732?

Yes, through Translation Shims for a new Networking API or Edit/Rewrite your software to support the interface.

3.1.4.6.1.3 What is the impact of the Networking API Service Definition approach on the current SCA definition?

1. Does the Networking API approach change the definition of any existing interfaces? *No.*

2. How many new interfaces are created by the Networking API approach? *The number of new interfaces can vary for each different Networking API Service Definition. Any new interfaces will be defined outside the Core Framework and thus "bypass the core framework." Since this Networking API approach can use the Core Framework interfaces "as is" or defined new interfaces "outside" the Core Framework, this Networking API approach does not impact the Core Framework*

3. What is the impact of the Networking API approach on the Core Framework? *The Core Framework will have to be able to track Networking API Service Definition GUIDs and version numbers in resource profiles so that it can instantiate resources that can interoperate.*

3.1.4.6.2 Able to support new networking and non-networking waveforms

Yes, by modifying already developed Networking API Service Definitions and changing the version number of creating a new Networking API Service Definition and getting a new GUID.

3.1.4.6.2.1 Does the Networking API Service Definition approach allow incorporation of extensions for new functionality?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which operations are used and what the name/value pairs are. Also, extensions are easily supported through the Networking API inheritance. This assumes that the controlling entity is longer Rooftop but has been transferred to some open standards body.

3.1.4.6.2.2 Does the Networking API Service Definition approach support future waveforms in a consistent manner (e.g. reuse of the Networking API for multiple future waveforms)?

Yes, through the use of the four primitives and inheritance.

3.1.4.6.2.3 Does the Networking API Service Definition approach support creation of Networking APIs that can be used by networking and non-networking waveforms?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which primitives are used and what the name/value pair variables are.

3.1.4.6.2.4 Is there a projected waveform that cannot be supported by the Networking API Service Definition approach?

None known. It is assumed that the approach can be extend and add new commands and variables and does not include unneeded commands and variables. The API has been examined for a wide variety of networking waveforms under the GloMo and other programs, including Aspen, Sparrow, Wideband Network Radio, etc.

3.1.4.6.3 Able to support legacy networking and non-networking waveforms

Yes, by modifying already developed Networking API Service Definitions and changing the version number of creating a new Networking API Service Definition and getting a new GUID.

3.1.4.6.3.1 Does the Networking API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking API?

This service definition approach has more overhead for sending control (due to the parsing of the name/value pairs) that are passed above IDL than an approach that passes parameters is specific IDL without any name/value pairs.

3.1.4.6.3.2 Does the Networking API Service Definition approach allow incorporation of extensions for new functionality?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which operations are used and what the name/value pairs are. Also, extensions are easily supported through the Networking API inheritance. This assumes that the controlling entity is longer Rooftop but has been transferred to some open standards body.

3.1.4.6.3.3 Does the Networking API Service Definition support legacy networking waveforms in a consistent manner (e.g. reuse of the Networking API for multiple legacy waveforms)?

The Networking API provides a consistent set of primitives plus inheritance upon which to standardize the Networking API definitions for a particular waveform. Thus, it encourages consistency.

3.1.4.6.3.4 Does the Networking API Service Definition support creation of Networking APIs that can be used by networking and non-networking waveforms?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which primitives are used and what the name/value pair variables are.

3.1.4.6.3.5 Is there a known waveform that cannot be supported by the Networking API Service Definition approach?

None known. It is assumed that the approach can be extend and add new commands and variables and does not include unneeded commands and variables. The API has been examined for a wide variety of networking waveforms under the GloMo and other programs, including Aspen, Sparrow, Wideband Network Radio, etc.

3.1.4.6.4 Performance (Efficiency/Latency)

3.1.4.6.4.1 Does the Networking API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking API?

This service definition approach has more overhead for sending control (due to the parsing of the name/value pairs) that are passed above IDL than an approach that passes parameters is specific IDL without any name/value pairs.

3.1.4.6.4.2 Does the Networking API Service Definition approach support high data throughput, low latency control?

This service definition approach has more overhead for sending control (due to the parsing of the name/value pairs) that are passed above IDL than an approach that passes parameters is specific IDL without any name/value pairs.

3.1.4.6.4.3 How is performance affected by any required message parsing within the producer and consumer resources?

Performance would be less efficient using name/value pairs as compared to a "pure IDL" approach.

3.1.4.6.5 Extensibility

3.1.4.6.5.1 Does the Networking API Service Definition approach allow Networking API definition to be extended to support new waveform interfaces, functionality, representation, and mechanisms?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which primitives are used and what the name/value pair variables are.

3.1.4.6.5.2 Does a mechanism(s) exist to limit proliferation of published Networking API extensions?

This Networking API approach per se does not have any mechanisms to limit proliferation of published Networking API extensions, although the use of inheritance should help the extensions stay somewhat similar.

However, this approach assumes that there is a standard configuration control body that determines which published Networking API extensions are the "standard" Networking API for a given waveform. In that sense it is very similar to SNMP which will allow a plethora of MIB definitions, but has the IETF define which MIBs are the standard MIBs that all routers/hosts must/should support.

3.1.4.6.6 Networking API Reuse at same layer

3.1.4.6.6.1 Does the Networking API Service Definition approach foster Networking API reuse at the same layer, e.g. use the same modem Networking API definition for multiple (different) waveforms?

Yes, somewhat through the use of inheritance. Also the standard configuration control body could foster such reuse.

3.1.4.6.6.2 Does the Networking API Service Definition approach foster commonality of interface?

1. Common Operation Set? *Yes.*

2. Common Parameter Set? *Somewhat through the use of inheritance (although the standard configuration control body could foster such commonality).*

3. Common Message/Data Format? *Yes, through inheritance.*

3.1.4.6.7 Component Interoperability (within the box, component plug and play)

3.1.4.6.7.1 Does the Networking API Service Definition approach foster development of Networking APIs that allow individual waveform components to be reused by other waveforms?

Yes, through the use of inheritance although it has problems because it does not define behavior.

3.1.4.6.7.2 Do the Networking API definition, implementation, and configuration control methods support component interoperability?

1. Does the Networking API approach support interface negotiation (e.g. Global User ID)? *Yes.*

2. Does the Networking API approach support interface version query? *Yes.*

3. Does the Networking API approach support backward interface compatibility? *Yes, through GUID & version interface query and fallback although it has problems because it does not define behavior.*

3.1.4.6.8 Cost

3.1.4.6.8.1 Is the Networking API Service Definition a cost driver?

1. Implementation Cost? *Moderately higher than "pure IDL" due to second layer of name/value pair representations in the var groups, but much less than allowing GUID only and any functional & representation definition as long as it is documented and identified by the GUID. (Should be similar to that required to implement SNMP & SNMP MIBs. The cost could significantly drop with the introduction of name/value pair compilers and development tools similar to those used by SNMP.)*

2. Maintenance Cost? *Moderately higher than "pure IDL" due to second layer of name/value pair representations in the var groups, but much less than allowing GUID only and any functional & representation definition as long as it is documented and identified by the GUID. (Should be similar to that required to maintain SNMP & SNMP MIBs. The cost could significantly drop with the introduction of name/value pair compilers and development tools similar to those used by SNMP.)*

3. Porting Cost? *If talking with applications that already understand/support Networking API, then it should be the same to slightly higher than "pure IDL" due to second layer of name/value pair representations in the var groups. (Similar to how inexpensive it is to port SNMP applications & agents.)*

3.1.4.6.9 Commercial Acceptance

3.1.4.6.9.1 Does the Networking API Service Definition approach have current or projected commercial acceptance, (e.g. SDRF, IEEE, OMG, or de facto)? (No, some, or widespread commercial acceptance?)

Many wireless standards are being developed due to the rapid explosion in commercial wireless products. Although we do not know which of these commercial initiatives will achieve commercial de facto standard acceptance similar to the Hayes modem AT command set, several such interface de facto standards are bound to arise. The one that wins is probably going to be based around getting initial market penetration in both the commercial cellular data market, the wireless LAN market, and the consumer home/personal network market. Since this Networking API is being defined for JTRS which is focused on military applications, it is likely that neither this Networking API (nor any Networking APIs) will achieve widespread commercial acceptance. Although it is not likely that any JTRS Networking API will achieve some commercial acceptance (assuming there are no technical problems with the Networking API). Since the JTRS addresses a large, important military market (although small in comparison to the commercial market); any JTRS Networking API that does not have technical problems is likely to be accepted by the SDRF and/or the IEEE.

Since, this Networking API approach allows the reuse of existing commercial interfaces for particular waveforms; it can get some commercial acceptance just by tracking with existing

commercial acceptances. Note that the high flexibility of this approach is likely to be a major issue for a significant minority of engineers.

Thus, the combination of the exploding commercial wireless market coupled with the fact that this Networking API approach has not technical negatives means that this approach, if selected as the JTRS Networking API, would likely achieve some commercial acceptance and become a SDRF and/or IEEE standard.

3.1.4.6.9.2 Does the Networking API Service Definition foster the adoption of Networking APIs with current or projected commercial acceptance, (e.g. DLPI, Sockets, de facto (Microsoft))? (No, some, or widespread commercial acceptance?)

It is possible, but not as likely as the Commercial Model (with or without inheritance) This approach will not allow the incorporation of other standards such as DLPP.

3.1.4.6.10 Security

This section requires further work and coordination with the Security IPT.

3.1.4.6.10.1 Do the JTRS security requirements prevent the use of this Networking API definition or implementation?

1. Use of IDL Networking API definition if red and black objects can only communicate through INFOSEC adapters? *This is a question if all messages flowing from red to black must pass through the generic message interface of an INFOSEC adapter. How can IDL interface definitions be used between red and black objects?*

2.

A complete answer will await meeting with the Security IPT.

3.1.4.6.11 Ability to Adapt to Commercial Trends

3.1.4.6.11.1 Is the Networking API Service Definition approach able to adapt to emerging commercial networking standards?

Yes, it is able to adapt to emerging standards through it extensibility, although not as well as alternatives #3 and #4, since this alternative is already frozen into having to use its primitives.

3.1.5 Option 4 - GloMo Rooftop API based Networking API Service Definition

Interface Type: Networking API Service Definition

Name: Option 5 – GloMo Rooftop API based Networking API Service Definition

Date: 21 March 2000

Revision: 3.0

Author: Jim Stevens (jasteven@collins.rockwell.com)

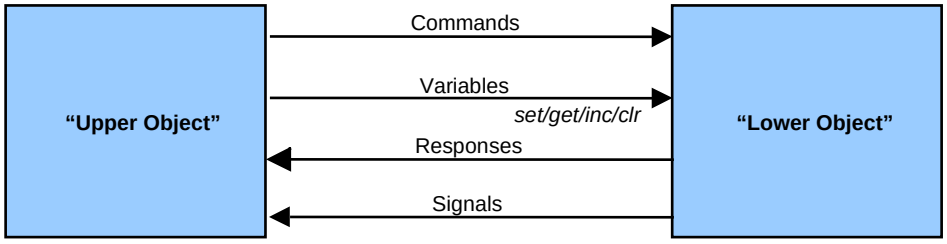
Company: Rockwell Collins

3.1.5.1 Description

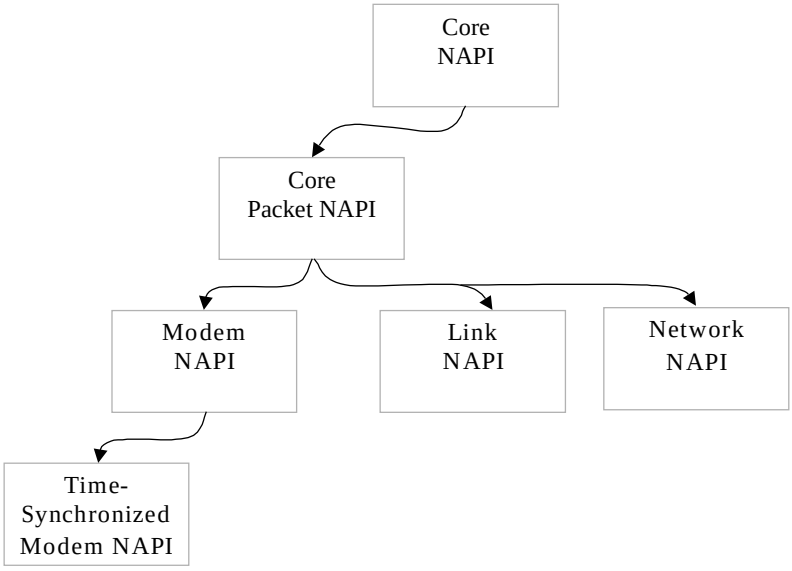
This Networking API Service Definition approach is based upon the GloMo Rooftop API framework except that the primitives are specified in IDL. This is the Networking API that the MSRC proposed during JTRS Phase 1. Since this service definition includes inheritance, it can be considered a special instance of Alternative Option 4 Networking API Service Definition.

3.1.5.2 Diagram

The following diagram shows the four primitives derived from the GloMo Rooftop API.



The following diagram shows an example of the Networking API inheritance.



3.1.5.3 Theory of Operation

This Networking API is based upon the GloMo Rooftop API. For more information on the current GloMo APIs, refer to the GloMo API Framework and Definition documents available at http://www.rooftop.com/technologyFRAME.html.

Each Networking API defined under this approach consists of the following components:

"Primitives"—used to define the basic information flow across the Networking API,

"Qualifiers"—flags and other action specifiers applicable to any of the primitives, and

"Return Codes"—status codes returned from certain primitive operations.

The four basic types of Networking API primitives are described below and illustrated in the first diagram in section 2.

Commands	Commands are asynchronous upper-to-lower object primitives for performing immediate, typically non-persistent actions. Commands often result in an immediate Response followed later by one or more Signals from the lower object.
Variables	Control characteristics and measurement status information of the lower object are communicated using Variable primitives. Variables support one or more of the set, get, increment, or clear synchronous access operations. Also, "variable groups" allow for the upper object to access a group of variables with one operation. Note that variables are defined as name/value pairs.

Responses	Responses report the synchronous lower-object result to an upper object's command or variable operation. For Commands, the Response often indicates whether or not the Command has been received correctly and can be acted upon, with one or more Signals reporting the result of the action later.
Signals	Signals are asynchronous lower-to-upper object primitives for reporting recent, typically non-persistent events. The lower object should support the selective enabling and disabling of each of its supported signals through the Networking API.

Each primitive can be qualified to give more specific instructions such as specifying which "radio transceiver" or specifying to which section (e.g., xmt or rcv) the operation should be applied. A special "info" qualifier is used with variable operations to allow the upper object to learn the capabilities (e.g., read/write support, range of valid values, default value) of the lower object with regards to a particular variable. Each Networking API defines a set of return codes to provide a standard means for the lower object to indicate the success or failure status in each Response to Command and Variable operations, and in each asynchronous Signal delivered to the upper object.

The Networking API Framework also defines a basic set of "Core" primitives that are applicable to all Networking APIs defined within this framework. This set with additional "Core Packet" primitives applicable to all packet-based Networking APIs (such as the Modem, Link, and Network Networking APIs). These sets are inheritable to develop additional Networking APIs such as the Modem, Link, and Network Networking APIs that are unique for each waveform implemented in JTRS.

The Networking APIs and inherited Networking APIs will be identified by GUIDs. There will be a controlling body for GUIDs that controls the control assignment of GUIDs. This controlling will assign a status for each Networking API instantiation. Potential statuses could include:

mandatory (i.e. if implementing a particular waveform, then must implement at least this interface)

recommended (i.e. if implementing a particular waveform, then recommend that implement this interface)

allowable (i.e. if implementing a particular waveform, then it is okay to implement this interface)

Other possible statuses including experimental, not recommended, etc. recommended

Any Networking APIs that are chosen as JTRS standard interfaces (that is mandatory or recommended) will be published and non-proprietary. However, vendors can choose to keep the details of particular Networking APIs private or proprietary (so that the status is likely to be allowable or some other.) The owner of interface controls version numbering. This is similar to the approach used to managing MIBs in SNMP and CMIP.

All visible interfaces will be uniquely identified. A mechanism will be provided within the Domain/Resource Manager to identify the interfaces used by the resources. When the Domain/Resource Manager instantiates a Resource to satisfy a User request it selects the resource with matching interface.

3.1.5.4 Operational Environment (OE)

This approach requires support within the OE to help negotiate interfaces during resource instantiation.

The openness of this approach means that some Networking API mechanisms may be chosen that lie outside the approach used by the OE (i.e. do not use the message interface).

3.1.5.5 Protocol Layer Applicability

The GloMo API has already been applied to the Modem, Link, and Network protocol layers and should be extensible to other layers (such as Transport) as needed.

3.1.5.6 Networking API FOM Analysis

3.1.5.6.1 Waveform Portability

Does the Networking API Service Definition approach foster waveform portability?

Yes, by developing a small number of primitives and a framework based upon IDL for encoding the service definition, the Networking API Service Definition can be moved to different platforms.

Note: It is assumed that 'waveform portability' means porting a complete waveform (and no hardware) from one JTRS platform to another, and not mixing and matching waveform components.

3.1.5.6.1.1 Identify the porting steps required to take a waveform to another JTRS Platform and evaluate the impact that the Networking API Service Definition approach has on that process.

3.1.5.6.1.1.1 Porting Classification

The following table lists the projected porting steps for a wide range of porting stimulus.

Porting Stimulus	**Description**	***Projected Porting Steps Required Because of the Networking API Service Definition***
Environment Change: Different Processor	Waveform must be ported to a different processor	*No Networking API effect.*
Environment Change: Different OS	Waveform must be moved to a different OS	*No Networking API effect.*
Environment Change: Different Programming Language	Waveform Objects must interact with objects on the new platform that are written in a different language	*No Networking API effect.*
Environment Change: Different Platform Hardware	Waveform is hosted on new platform hardware that may require additional or different control, e.g. modem, INFSOEC, red I/O	*No Networking API effect.*
Environment Change: Different Transport Mechanism	Waveform is hosted on a new platform that uses a different transport mechanism than the source platform	*Direct Rehost* ***or*** *Translation Shim* ***depending on the transport mechanism.***

The projected porting steps are defined from least to most effort in the following table.

Porting Step	Description	Effort Level
Direct Rehost	Same processors, OSs etc. – No changes required	None
Recompile	Recompile of those portions of the waveform that are hosted on a different processor	Minimal
Translation Shim	Does the Networking API approach require that shims be added to the waveform to port to the new platform?	Moderate
Edit/ReWrite	Does the Networking API approach require such drastic code changes that edit/rewrite is necessary?	Moderate-Extensive

3.1.5.6.1.2 Does the Networking API Service Definition foster waveform portability for software that already supports an existing interface, e.g. ARINC 732?

Yes, through Translation Shims for a new Networking API or Edit/Rewrite your software to support the interface.

3.1.5.6.1.3 What is the impact of the Networking API Service Definition approach on the current SCA definition?

1. Does the Networking API approach change the definition of any existing interfaces?

No.

2. How many new interfaces are created by the Networking API approach?

The number of new interfaces can vary for each different Networking API Service Definition. Any new interfaces will be defined outside the Core Framework and thus "bypass the core framework." Since this Networking API approach can use the Core Framework interfaces "as is" or defined new interfaces "outside" the Core Framework, this Networking API approach does not impact the Core Framework.

3. What is the impact of the Networking API approach on the Core Framework?

The Core Framework will have to be able to track Networking API Service Definition GUIDs and version numbers in resource profiles so that it can instantiate resources that can interoperate.

3.1.5.6.2 Able to support new networking and non-networking waveforms

Yes, by modifying already developed Networking API Service Definitions and changing the version number of creating a new Networking API Service Definition and getting a new GUID.

3.1.5.6.2.1 Does the Networking API Service Definition approach allow incorporation of extensions for new functionality?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which operations are used and what the name/value pairs are. Also, extensions are easily supported through the Networking API inheritance. This assumes that the controlling entity is longer Rooftop but has been transferred to some open standards body.

3.1.5.6.2.2 Does the Networking API Service Definition approach support future waveforms in a consistent manner (e.g. reuse of the Networking API for multiple future waveforms)?

Yes, through the use of the four primitives and inheritance.

3.1.5.6.2.3 Does the Networking API Service Definition approach support creation of Networking APIs that can be used by networking and non-networking waveforms?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which primitives are used and what the name/value pair variables are.

3.1.5.6.2.4 Is there a projected waveform that cannot be supported by the Networking API Service Definition approach?

None known. It is assumed that the approach can be extend and add new commands and variables and does not include unneeded commands and variables. The API has been examined for a wide variety of networking waveforms under the GloMo and other programs, including Aspen, Sparrow, Wideband Network Radio, etc.

3.1.5.6.3 Able to support legacy networking and non-networking waveforms

Yes, by modifying already developed Networking API Service Definitions and changing the version number of creating a new Networking API Service Definition and getting a new GUID.

3.1.5.6.3.1 Does the Networking API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking API?

This service definition approach has more overhead for sending control (due to the parsing of the name/value pairs) that are passed above IDL than an approach that passes parameters is specific IDL without any name/value pairs.

3.1.5.6.3.2 Does the Networking API Service Definition approach allow incorporation of extensions for new functionality?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which operations are used and what the name/value pairs are. Also, extensions are easily supported through the Networking API inheritance. This assumes that the controlling entity is longer Rooftop but has been transferred to some open standards body.

3.1.5.6.3.3 Does the Networking API Service Definition support legacy networking waveforms in a consistent manner (e.g. reuse of the Networking API for multiple legacy waveforms)?

The Networking API provides a consistent set of primitives plus inheritance upon which to standardize the Networking API definitions for a particular waveform. Thus, it encourages consistency.

3.1.5.6.3.4 Does the Networking API Service Definition support creation of Networking APIs that can be used by networking and non-networking waveforms?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which primitives are used and what the name/value pair variables are.

3.1.5.6.3.5 Is there a known waveform that cannot be supported by the Networking API Service Definition approach?

None known. It is assumed that the approach can be extend and add new commands and variables and does not include unneeded commands and variables. The API has been examined for a wide variety of networking waveforms under the GloMo and other programs, including Aspen, Sparrow, Wideband Network Radio, etc.

3.1.5.6.4 Performance (Efficiency/Latency)

3.1.5.6.4.1 Does the Networking API Service Definition approach support low-overhead, efficient transfer of control and traffic across the Networking API?

This service definition approach has more overhead for sending control (due to the parsing of the name/value pairs) that are passed above IDL than an approach that passes parameters is specific IDL without any name/value pairs.

3.1.5.6.4.2 Does the Networking API Service Definition approach support high data throughput, low latency control?

This service definition approach has more overhead for sending control (due to the parsing of the name/value pairs) that are passed above IDL than an approach that passes parameters is specific IDL without any name/value pairs.

3.1.5.6.4.3 How is performance affected by any required message parsing within the producer and consumer resources?

Performance would be less efficient using name/value pairs as compared to a "pure IDL" approach.

3.1.5.6.5 Extensibility

3.1.5.6.5.1 Does the Networking API Service Definition approach allow Networking API definition to be extended to support new waveform interfaces, functionality, representation, and mechanisms?

Yes, by creating a new waveform definition or extension with its own globally unique ID that describes which primitives are used and what the name/value pair variables are.

3.1.5.6.5.2 Does a mechanism(s) exist to limit proliferation of published Networking API extensions?

This Networking API approach per se does not have any mechanisms to limit proliferation of published Networking API extensions, although the use of inheritance should help the extensions stay somewhat similar.

However, this approach assumes that there is a standard configuration control body that determines which published Networking API extensions are the "standard" Networking API for a given waveform. In that sense it is very similar to SNMP which will allow a plethora of MIB definitions, but has the IETF define which MIBs are the standard MIBs that all routers/hosts must/should support.

3.1.5.6.6 Networking API Reuse at same layer

3.1.5.6.6.1 Does the Networking API Service Definition approach foster Networking API reuse at the same layer, e.g. use the same modem Networking API definition for multiple (different) waveforms?

Yes, somewhat through the use of inheritance. Also the standard configuration control body could foster such reuse.

3.1.5.6.6.2 Does the Networking API Service Definition approach foster commonality of interface?

1. Common Operation Set? *Yes.*

2. Common Parameter Set? *Somewhat through the use of inheritance (although the standard configuration control body could foster such commonality).*

3. Common Message/Data Format? *Yes, through inheritance.*

3.1.5.6.7 Component Interoperability (within the box, component plug and play)

3.1.5.6.7.1 Does the Networking API Service Definition approach foster development of Networking APIs that allow individual waveform components to be reused by other waveforms?

Yes, through the use of inheritance although it has problems because it does not define behavior.

3.1.5.6.7.2 Do the Networking API definition, implementation, and configuration control methods support component interoperability?

1. Does the Networking API approach support interface negotiation (e.g. Global User ID)? *Yes.*

2. Does the Networking API approach support interface version query? *Yes.*

3. Does the Networking API approach support backward interface compatibility? *Yes, through GUID & version interface query and fallback although it has problems because it does not define behavior.*

3.1.5.6.8 Cost

3.1.5.6.8.1 Is the Networking API Service Definition a cost driver?

1. Implementation Cost?

Moderately higher than "pure IDL" due to second layer of name/value pair representations in the var groups, but much less than allowing GUID only and any functional & representation definition as long as it is documented and identified by the GUID. (Should be similar to that required to implement SNMP & SNMP MIBs. The cost could significantly drop with the introduction of name/value pair compilers and development tools similar to those used by SNMP.)

2. Maintenance Cost?

Moderately higher than "pure IDL" due to second layer of name/value pair representations in the var groups, but much less than allowing GUID only and any functional & representation definition as long as it is documented and identified by the GUID. (Should be similar to that required to maintain SNMP & SNMP MIBs. The cost could significantly drop with the introduction of name/value pair compilers and development tools similar to those used by SNMP.)

3. Porting Cost?

If talking with applications that already understand/support Networking API, then it should be the same to slightly higher than "pure IDL" due to second layer of name/value pair representations in the var groups. (Similar to how inexpensive it is to port SNMP applications & agents.)

3.1.5.6.9 Commercial Acceptance

3.1.5.6.9.1 Does the Networking API Service Definition approach have current or projected commercial acceptance, (e.g. SDRF, IEEE, OMG, or de facto)? (No, some, or widespread commercial acceptance?)

Many wireless standards are being developed due to the rapid explosion in commercial wireless products. Although we do not know which of these commercial initiatives will achieve commercial de facto standard acceptance similar to the Hayes modem AT command set, several such interface de facto standards are bound to arise. The one that wins is probably going to be based around getting initial market penetration in both the commercial cellular data market, the wireless LAN market, and the consumer home/personal network market. Since this Networking API is being defined for JTRS which is focused on military applications, it is likely that neither this Networking API (nor any Networking APIs) will achieve widespread commercial acceptance. Although it is not likely that any JTRS Networking API will achieve some commercial acceptance (assuming there are no technical problems with the Networking API). Since the JTRS addresses a large, important military market (although small in comparison to the commercial market); any JTRS Networking API that does not have technical problems is likely to be accepted by the SDRF and/or the IEEE.

Since, this Networking API approach allows the reuse of existing commercial interfaces for particular waveforms; it can get some commercial acceptance just by tracking with existing commercial acceptances. Note that the high flexibility of this approach is likely to be a major issue for a significant minority of engineers.

Thus, the combination of the exploding commercial wireless market coupled with the fact that this Networking API approach has not technical negatives means that this approach, if selected as the JTRS Networking API, would likely achieve some commercial acceptance and become a SDRF and/or IEEE standard.

3.1.5.6.9.2 Does the Networking API Service Definition foster the adoption of Networking APIs with current or projected commercial acceptance, (e.g. DLPI, Sockets, de facto (Microsoft))? (No, some, or widespread commercial acceptance?)

It is possible, but not as likely as the Commercial Model (with or without inheritance) This approach will not allow the incorporation of other standards such as DLPP.

3.1.5.6.10 Security

(This section requires further work and coordination with the Security IPT.)

3.1.5.6.10.1 Do the JTRS security requirements prevent the use of this Networking API definition or implementation?

1. Use of IDL Networking API definition if red and black objects can only communicate through INFOSEC adapters?

This is a question if all messages flowing from red to black must pass through the generic message interface of an INFOSEC adapter. How can IDL interface definitions be used between red and black objects?

2.

A complete answer will await meeting with the Security IPT.

3.1.5.6.11 Ability to Adapt to Commercial Trends

3.1.5.6.11.1 Is the Networking API Service Definition approach able to adapt to emerging commercial networking standards?

Yes, it is able to adapt to emerging standards through it extensibility, although not as well as alternatives #3 and #4, since this alternative is already frozen into having to use its primitives.

3.2 NETWORKING API TRANSFER MECHANISM OPTIONS

3.2.1 Option 1 - CORBA, CORBA, CORBA, CORBA

Interface Type: Transfer Mechanism

Name: CORBA, CORBA, CORBA, CORBA

Date: 20 MAR 00

Revision: 5.0

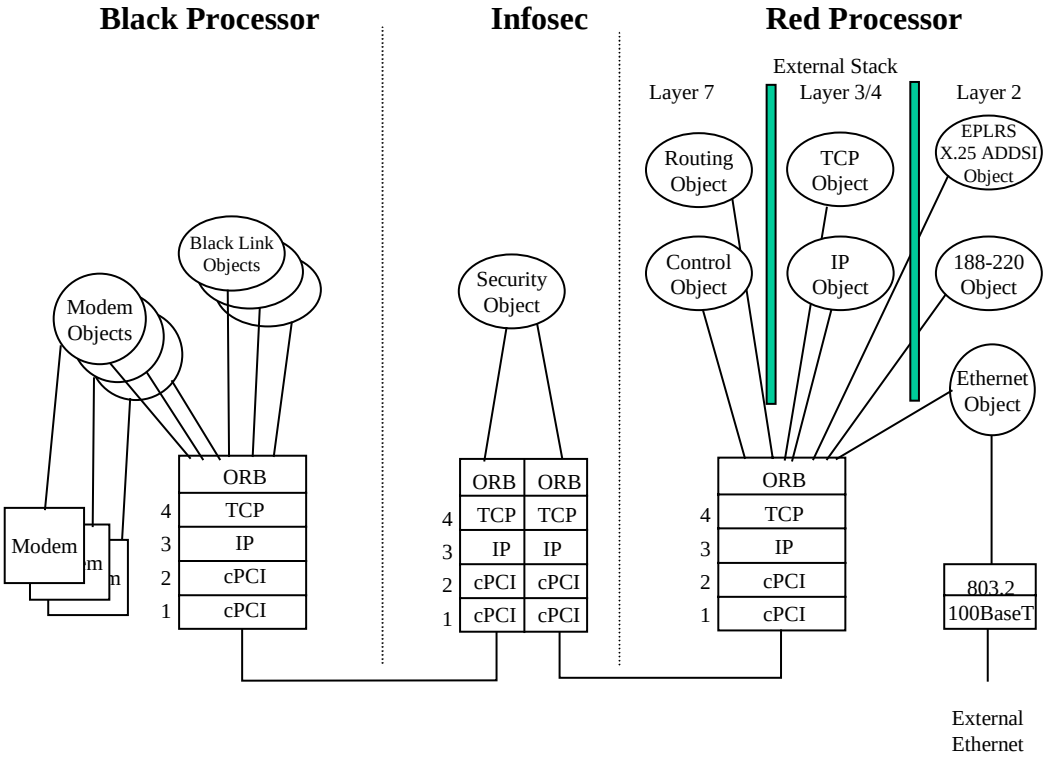
Author: Fred Mangarelli

Company: Raytheon

3.2.1.1 Description

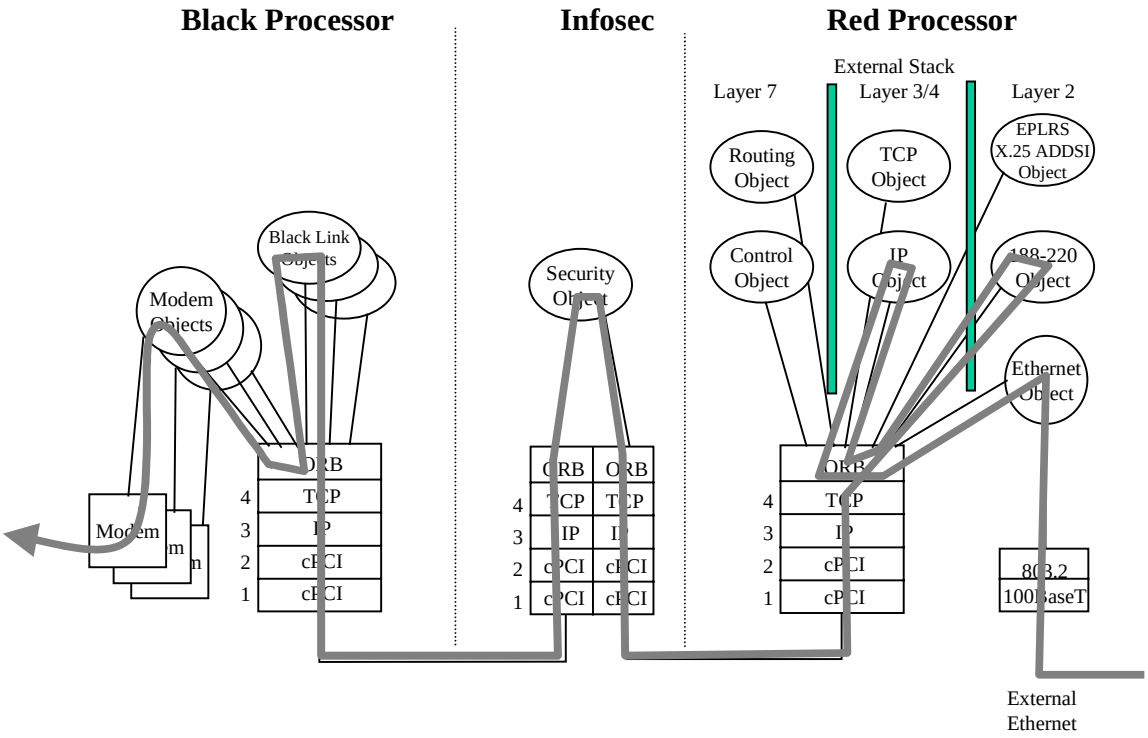
This alternative uses CORBA for non-real-time setup of waveform and non-real-time setting of operation parameters, Waveform Control across Red-to-Black Boundary, real-time control, and Waveform Data Flow across the Red-to-Black Boundary.

3.2.1.2 Diagram



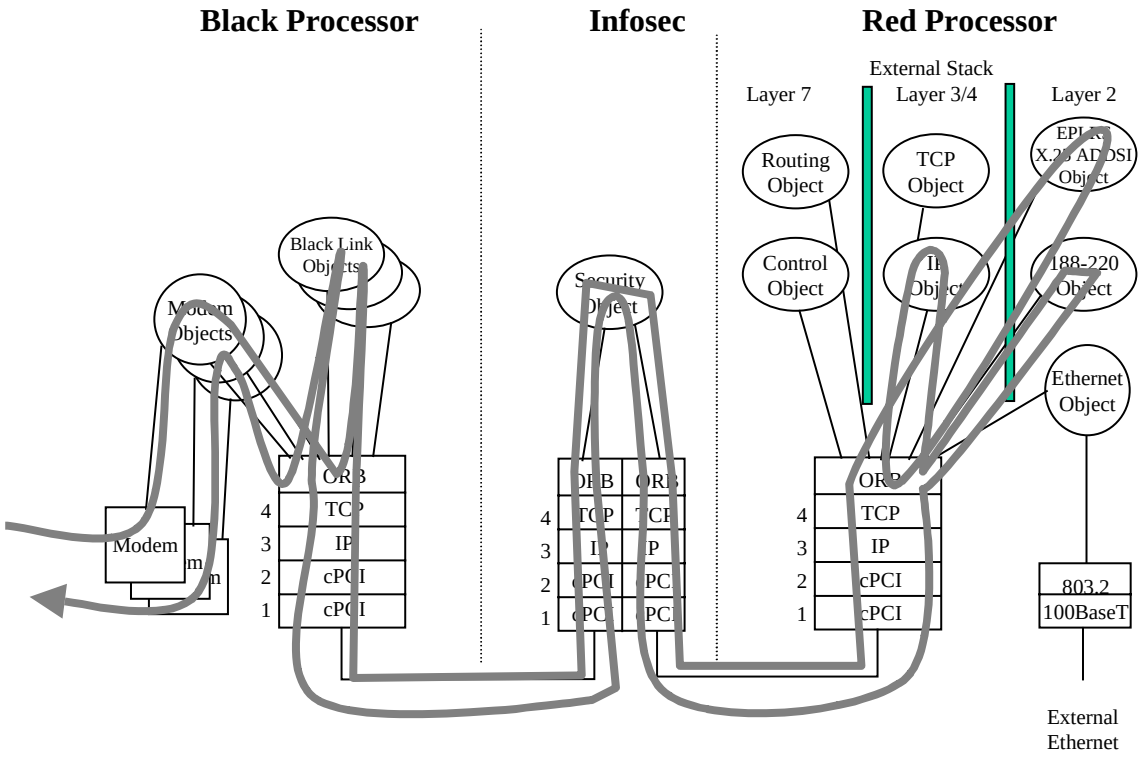
Data flow – Host packet on ethernet interface to be routed out of the SINCGARS (188-220) interface:

The packet is received on the ethernet interface and passed to the ethernet object. The ethernet object uses CORBA to pass the packet to the IP object, which uses CORBA to pass the packet to the 188-220 object. The 188-220 object passes the packet to the INFOSEC object and on to the black link object in the black processor. The black link passes the packet to the modem object where it is sent to the modem for RF output.



Data flow – Packet received on EPLRS interface to be routed out of the SINCGARS (188-220) interface:

The packet is received by the EPLRS modem and sent to the modem objects which uses CORBA to pass the packet to the black link object. The black link object passes the packet to the security object and then to the EPLRS X.25 ADDSI object. The EPLRS X.25 ADDSI object uses CORBA to pass the packet to the IP object, which uses CORBA to pass the packet to the 188-220 object. The 188-220 object passes the packet to the infosec object and on to the black link object in the black processor. The black link passes the packet to the modem object where it is sent to the modem for RF output.



3.2.1.3 Theory of Operation

- Supports use of COTS networking software such as Internet Protocol and associated routing and control protocols, TCP, UDP, SNMP, Web Servers, etc.
- Supports use of legacy tactical networking software such as MIL-STD-188-220, NTDR.
- Adapts COTS and legacy software into the JTRS Operating Environment through architecture compliant adapter objects.
- Uses architecture compliant adapter objects to support non-CORBA capable hardware such as INFOSEC cryptographic engine, modem and audio I/O.
- Uses CORBA transport for inter-object communications and object control by the JTRS Core Framework.
- All waveform dependent objects are at the Applications Layer for maximum waveform portability.
- Protocol stacks provide communications support for CORBA ORBs provided by TCP/IP.
- Separate internal and external stacks to isolate internal radio IP address space from external IP address space.
- This is an SCA compliant, highly portable application design.

3.2.1.4 OE

This approach uses the transport services provided by the OE for transport of information between CORBA Objects. An additional “external stack” is provided where each protocol layer is implemented as a CORBA object.

3.2.1.5 Layer Applicability

Modem, Link, Network (3A, 3B), Transport

3.2.1.6 FOM Analysis

3.2.1.6.1 Waveform Portability

Waveform portability is achieved with CORBA by developing each layer of the waveform protocol stack as a CORBA object. IDL is used to define the interface between stack layers. Applications communicate with one another regardless of the environment in which the application resides.

3.2.1.6.1.1 Porting Activity Identification

This section identifies the porting steps required for a given transfer mechanism alternative to be ported given certain stimuli.

3.2.1.6.1.1.1 Porting Stimuli Definitions

Stimulus	Description	Classification
Environment Change:Different Processor	Waveform must be ported to a different processor	Non-recurring
Environment Change:Different OS	Waveform must be moved to a different OS	Non-recurring
Environment Change:Different Bus	Waveform must be moved to a different bus structure	Non-recurring
Protocol Standard Migration:COTS Stack	Waveform requires new capabilities as reflected in new protocol standards (e.g. implementation of IP RFCs). The networking implementation uses COTS stack components.	Non-recurring
Protocol Standard Migration:Custom Stack	Waveform requires new capabilities as reflected in new protocol standards (e.g. implementation of IP RFCs). The networking implementation uses custom stack components.	Non-recurring
New Transport Protocol: COTS Stack	A new transport protocol must be added (e.g . to support reliable multicast). The networking implementation uses COTS stack components.	Non-recurring
New Transport Protocol: Custom Stack	A new transport protocol must be added (e.g . to support reliable multicast). The networking implementation uses custom stack components	Non-recurring

3.2.1.6.1.1.2 Porting Step Definitions

Porting Step	Description	Effort Level
Recompilation	Recompilation of networking part of waveform.	Minimal
Thin Layer Port	Port of OS/Kernel isolation layer	Moderate
Bus Layer Port	Port of bus communication layer	Moderate
Mechanism Port	Port of networking functionality to the chosen implementation mechanism (i.e. CORBA, STREAMS. Etc)	Extensive
Application Port	Implementation in a particular environment requires porting of one or more applications. For example a custom socket implementation may invalidate the use of select() in applications.	Minimal-Extensive
Capability Extension	Requires the extension of existing software to add new capabilities	Moderate-Extensive
OS Rebuild	Requires rebuilding the OS.	Minimal

3.2.1.6.1.1.3 Porting Steps Required for a Given Stimulus

Environment Change (non-recurring)			Protocol Standard Migration (non-recurring)		New Transport Protocol (non-recurring)	
Different Processor	**Different OS**	**Different Bus**	**COTS Stack**	**Custom Stack**	**COTS Stack**	**Custom Stack**
Recompilation	*Recompilation*	*No impact*	*Thin Layer Port*	*Capability Extension*	*Recompile or Capability Extension*	*Recompile or Capability Extension*

3.2.1.6.1.2 Impacts to CF caused by using the transfer mechanism.

There are no impacts to the CF.

3.2.1.6.2 Ability to support legacy networking

3.2.1.6.2.1 Ability to support legacy and future waveforms

Adapters/wrappers are used to adapt non-CORBA capable legacy code to CORBA transport.

Since CORBA is used as the transport between all protocol layers, overhead and performance is a factor in support of high-throughput/low-latency legacy and future waveforms.

3.2.1.6.2.2 Show how transport will support existing networking waveforms.

CORBA is able to support legacy networking waveforms such as SINCGARS SIP and EPLRS. For SINCGARS SIP, the IP network layer (3b) (shared by all network waveforms), the 188-220 intranet layer(3a), the link layer (2), the INFOSEC interface, the black link layer and the modem driver would each be implemented as CORBA objects. In the case of EPLRS the IP network layer, the X.25 ADDSI red link layer, the INFOSEC interface, the black side link layer and the modem driver would be implemented as CORBA objects. The interfaces between each of these objects would be represented in IDL.

Since CORBA is used as the transport between all protocol layers, overhead and performance is a factor in support of high-throughput/low-latency legacy and future waveforms.

3.2.1.6.2.3 Show how transport will support existing non-networking waveforms.

Use adapters/wrappers for non-CORBA capable hardware drivers (e.g. audio and modem adapters).

3.2.1.6.2.4 Show how transport will support future networking waveforms.

For future networking waveforms, the protocol layers are implemented as CORBA objects.

Since CORBA is used as the transport between all protocol layers, overhead and performance is a factor in support of high-throughput/low-latency future waveforms.

3.2.1.6.2.5 Show how transport will support future non-networking waveforms.

For future non-networking waveforms, the waveforms would be implemented as CORBA objects.

Since CORBA is used as the transport, overhead and performance is a factor in support of high-throughput/low-latency future non-networking waveforms.

3.2.1.6.3 Performance

Performance is the ability of the transport mechanism to support the message data rate, timing and latency requirements of the modem hardware. It is difficult to quantify performance when the characteristics of the modem hardware, the processor, the operating system and the CORBA implementation are not known.

Modem hardware: *There is a tradeoff between hardware cost and complexity and real time software performance. The modem hardware to a certain extent, can be designed to compensate for software latency by providing buffering of commands and data. For example, a modem with a time slot architecture, such as JTIDS, can be designed with buffering for multiple time slots. This would allow the protocol software to set up time slot parameters in advance, making the software control less time critical.*

Processor: *Processor speed plays an important role in real time performance. The processing burdens of using CORBA as a transport mechanism can be*

offset to a certain extent by a fast processor. If too much reliance is placed on processing speed, there is a risk that the protocol stack will perform on all versions of the JTRS hardware. The portability aspects of CORBA become limited if the software can only run on certain processors.

Operating System: *Operating systems also play an important role in the performance of real time systems. Every aspect of the operating system, from scheduling to interrupt processing can have an impact on the real time performance of a system. Some modems may require their protocol software to run in a real time operating system, such as VxWorks or pSOS in order to meet performance requirements.*

CORBA Implementation: *The specific implementation of the CORBA middleware also has an impact on the real time performance of the system. CORBA implementation issues include allowing a pass by reference between objects in the same memory space, the specific transport mechanism used to communicate between objects in the same processor space, and the extent to which marshalling is performed. Performance considerations for certain modems may force a specific implementation or CORBA, which would limit the object portability.*

3.2.1.6.3.1 Identify processing steps for a packet from input to output of a protocol stack

3.2.1.6.3.1.1 Unicast Packet from Ethernet to Modem

Layers and Layer Transitions	**Quantums**
Ethernet Link Layer	*allocate packet buffer + strip ethernet header + convert packet to CORBA object*
Transition to IP Network Layer	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
IP Network Layer	*No additional overhead*
Transition to Link Module (same processor as network module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Transition to Link Module (different processor from network module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Link Module (DLPI)	*No additional overhead*
Transition to Modem Module (same processor as link module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Transition to Modem Module (different processor from link module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Modem Module	*context switch + data copy (pass by value) + buffer deallocation*

3.2.1.6.3.1.2 Multicast Packet from Ethernet to Modem (3 channels)

Layers and Layer Transitions	**Quantums**
Ethernet Link Layer	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Transition to IP Network Layer	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
IP Network Layer	*No additional overhead*
Transition to Link Module (same processor as network module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Transition to Link Module (different processor from network module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead * 3*
Link Module (DLPI)	*No additional overhead*
Transition to Modem Module (same processor as link module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead * 3*
Transition to Modem Module (different processor from link module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead * 3*
Modem Module	*context switch + data copy (pass by value) + buffer deallocation * 3*

3.2.1.6.3.1.3 Message from Application to Modem using Sockets and TCP

Layers and Layer Transitions	Quantums
Stream Head	N/A
Transition to Sockets Module	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Sockets Module	*No additional overhead*
Transition to TCP Module	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
TCP Module (TPI)	*No additional overhead*
Transition to IP Network Layer	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
IP Network Layer	*No additional overhead*
Transition to Link Module (same processor as network module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Transition to Link Module (different processor from network module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Link Module (same processor as network module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Transition to Modem Module (same processor as link module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Transition to Modem Module (different processor from link module)	*context switch + buffer allocation + data copy (pass by value) + buffer deallocation + transport mechanism overhead*
Modem Module	*context switch + data copy (pass by value) + buffer deallocation*

3.2.1.6.3.2 Identify possible QoS impacts

- *Possible non-deterministic behavior*
- *Possibly excessive latency*
- *Possibly excessive processing overhead*

3.2.1.6.4 Extensibility

3.2.1.6.4.1 Does the protocol implementation mechanism support addition of newly defined or extension to existing protocol layers?

Since this approach makes extensive use of CORBA transport between application objects, addition of newly defined protocol layers is easy, and changes in waveform implementation (addition of waveform objects) is also easy. Protocol layers can easily be inserted as objects. Since all objects must conform to the interface defined in IDL, new protocol objects can be written to conform to the existing IDL and inserted into the stack.

3.2.1.6.5 Construct/Deconstruct Protocol Stacks

3.2.1.6.5.1 Address ability to add/remove layers without OS rebuild

Fully supported. Since CORBA transport is used no OS rebuild is necessary to add/remove application layers. This is simply of function of waveform application instantiation/teardown.

3.2.1.6.5.2 Address ability to distribute protocol layers across address spaces.

This is fully supported.

3.2.1.6.5.3 Address dynamic construction/deconstruction of protocol stacks

Wherever CORBA transport is used this is simply a function of waveform application instantiation/teardown.

3.2.1.6.6 Component Interoperability

3.2.1.6.6.1 Address how components can be used between waveforms

With respect to the transport mechanism, use of CORBA transport between waveform objects supports/enables component reuse. The limiting factors become component functionality and interface definition, not transport.

3.2.1.6.6.2 Address how components interoperate across differing processors

Use of CORBA transport makes this transparent. CORBA Bypass adapters running on different processors on the red and black sides must be concerned with data representation/interpretation.

3.2.1.6.7 Cost

Identify cost drivers and attempt to quantify in high level terms.

3.2.1.6.7.1 Implementation Cost

- Level of custom development – *Limited to waveform objects*
- Labor – *Use of CORBA transport minimizes waveform development labor*
- Licenses - *None*
- COTS – *Use of CORBA transport through adapters allows maximum use of COTS.*
- *Use of CORBA transport allows use of "large" COTS components.*
- Integration – *Use of CORBA transport allows integration to concentrate on application integration*
- *and not on infrastructure integration.*

Potential cost of using & wrapping COTS software, such as IP stacks

3.2.1.6.7.2 Maintenance Cost

- Effect of version migration of protocol standards (e.g. IP).

As protocol standards evolve waveform applications may have to be updated. This is relatively uncomplicated when CORBA transport is used because effort is limited to application objects only.

3.2.1.6.7.3 Porting Cost

Use of CORBA transport yields high portability between platforms and minimizes porting costs.

3.2.1.6.8 Commercial Acceptance

3.2.1.6.8.1 Current Commercial Acceptance

Although CORBA has wide commercial acceptance as a mechanism for transparently passing information between objects, it is unlikely that CORBA will evolve as an implementation for protocol stack layers.

- No Commercial Acceptance <---------- *most likely case & current commercial case*
- Some Commercial Acceptance
- Widespread Commercial Acceptance

3.2.1.6.8.2 Future Commercial Acceptance

Unknown? It is considered unlikely.

3.2.1.6.9 Security

3.2.1.6.9.1 Can support Local/Global Address Space

Yes

3.2.1.6.9.2 Does transport meet security requirements?

Not known.

3.2.1.6.9.3 How does transport handle control/status, header and plain text bypass?

Don't know, but possible security issue. Note that Raytheon/Rooftop did this with PTAPI on WRN program (although this was not certified).

3.2.1.6.10 Ability to Adapt to Commercial Trends

Identify how the given transfer mechanism could support moving to other implementations.

Unknown. It is unlikely that CORBA will be used to implement protocol stacks in future implementations.

3.2.2 Option 2- CORBA, CORBA, Non-CORBA, Non-CORBA

Interface Type: Transfer Mechanism

Name: CORBA, CORBA, Non-CORBA transfer mechanism for Real-Time Waveform Control across Red-to-Black Boundary, Non-CORBA transfer mechanism for Waveform Data Flow across Red-to Black Boundary

Date: 20 MAR 00

Revision: 4.0

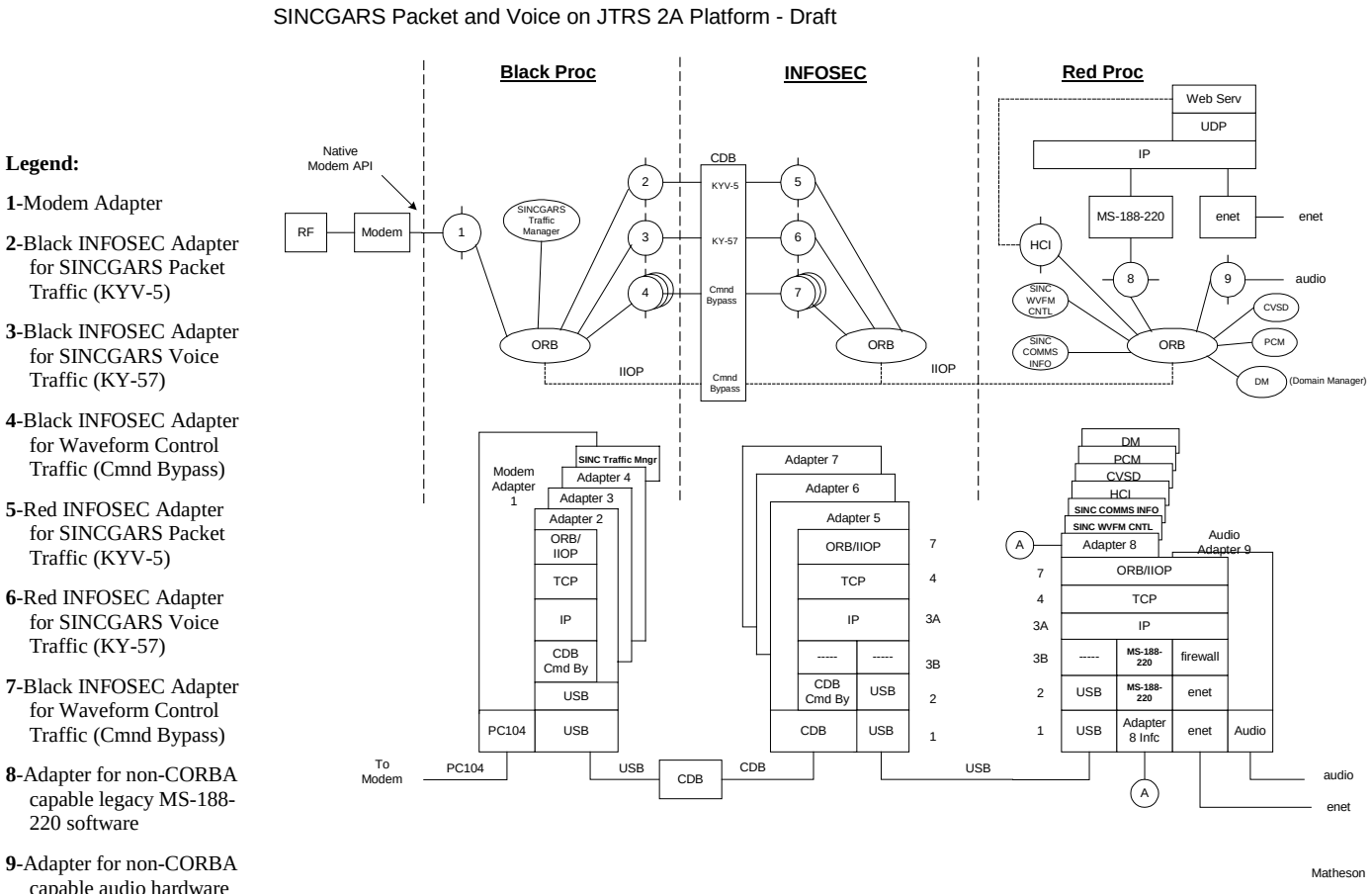
Author: L. Matheson

Company: ITT

3.2.2.1 Description

This alternative uses CORBA for non-real-time setup of waveform and non-real-time setting of operation parameters, Non-CORBA transfer mechanism for Waveform Control across Red-to-Black Boundary for real-time control, and Non-CORBA transfer mechanism for Waveform Data Flow across the Red-to-Black Boundary.

3.2.2.2 Diagram



Legend:

1-Modem Adapter

2-Black INFOSEC Adapter for SINCGARS Packet Traffic (KYV-5)

3-Black INFOSEC Adapter for SINCGARS Voice Traffic (KY-57)

4-Black INFOSEC Adapter for Waveform Control Traffic (Cmnd Bypass)

5-Red INFOSEC Adapter for SINCGARS Packet Traffic (KYV-5)

6-Red INFOSEC Adapter for SINCGARS Voice Traffic (KY-57)

7-Black INFOSEC Adapter for Waveform Control Traffic (Cmnd Bypass)

8-Adapter for non-CORBA capable legacy MS-188-220 software

9-Adapter for non-CORBA capable audio hardware

Figure 1: Representation of SINCGARS Waveform on JTRS 2A Platform

3.2.2.3 Theory of Operation

- Supports use of COTS networking software such as Internet Protocol and associated routing and control protocols, TCP, UDP, SNMP, Web Servers, etc.
- Supports use of legacy tactical networking software such as MIL-STD-188-220, NTDR.
- Adapts COTS and legacy software into the JTRS Operating Environment through architecture compliant adapter objects.
- Uses architecture compliant adapter objects to support non-CORBA capable hardware such as INFOSEC crypto engine, modem and audio I/O.
- Uses CORBA transfer mechanism for inter-object communications and object control by the JTRS Core Framework.
- All waveform dependent objects are at the Applications Layer for maximum waveform portability.
- Protocol stacks show communications support for CORBA ORBs provided by TCP/IP.
- Firewall used on Ethernet interface to isolate internal radio IP address space from external IP address space.

The diagrams on the following pages illustrate flow through a radio for the following types of traffic:

- Domain Manager Control of Black Side Objects
- Waveform Data Traffic
- Waveform Control Traffic
- Waveform Voice Traffic

SINCGARS Packet and Voice on JTRS 2A Platform - Domain Manager Control of Black Side Object

Legend:

1-Modem Adapter

2-Black INFOSEC Adapter for SINCGARS Packet Traffic (KYV-5)

3-Black INFOSEC Adapter for SINCGARS Voice Traffic (KY-57)

4-Black INFOSEC Adapter for Waveform Control Traffic (Cmnd Bypass)

5-Red INFOSEC Adapter for SINCGARS Packet Traffic (KYV-5)

6-Red INFOSEC Adapter for SINCGARS Voice Traffic (KY-57)

7-Black INFOSEC Adapter for Waveform Control Traffic (Cmnd Bypass)

8-Adapter for non-CORBA capable legacy MS-188-220 software

9-Adapter for non-CORBA capable audio hardware

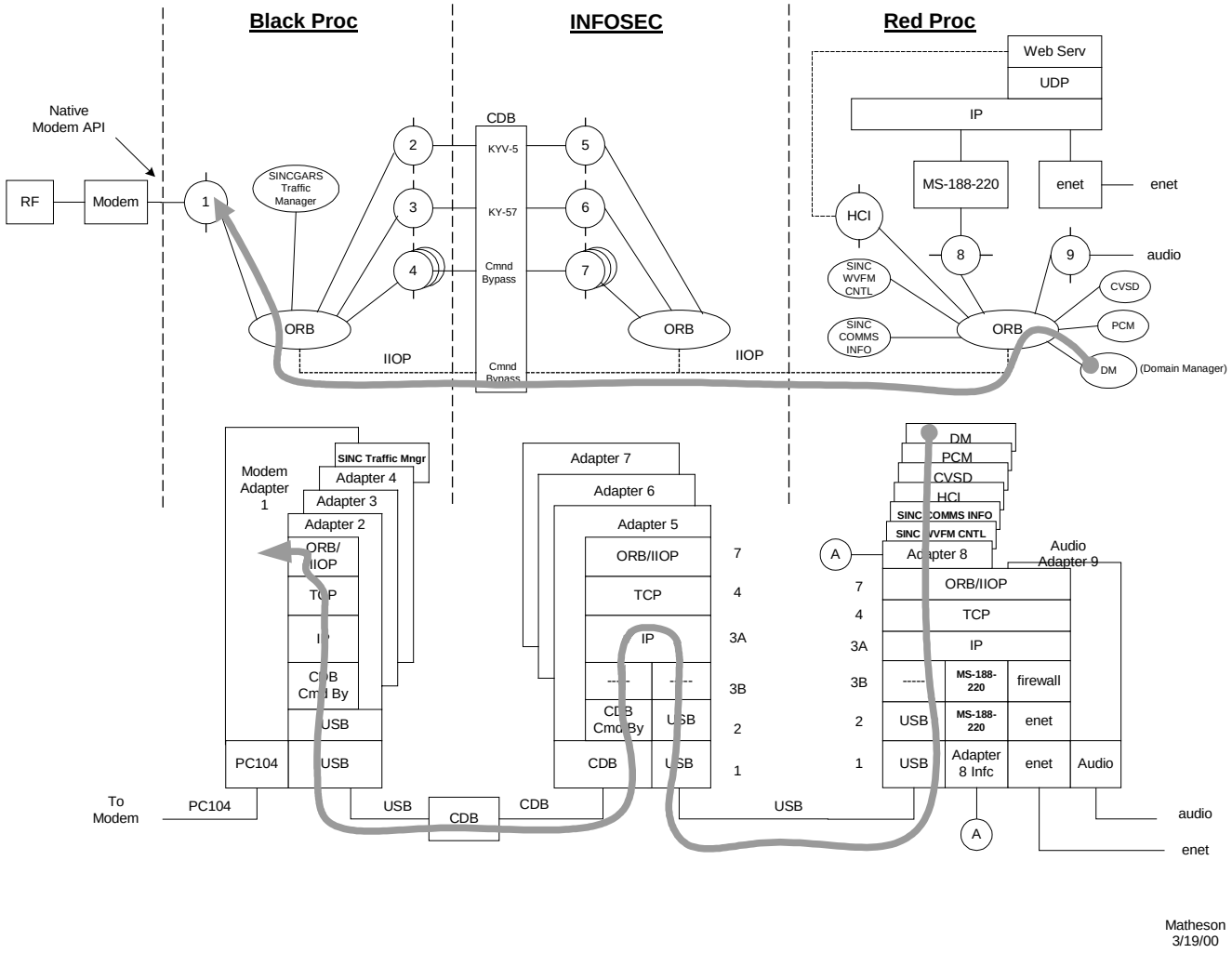


Figure 2: Domain Manager Control Traffic uses CORBA Transfer Mechanism only

SINCGARS Packet and Voice on JTRS 2A Platform - Input Unicast Ethernet Packet and Output on Modem

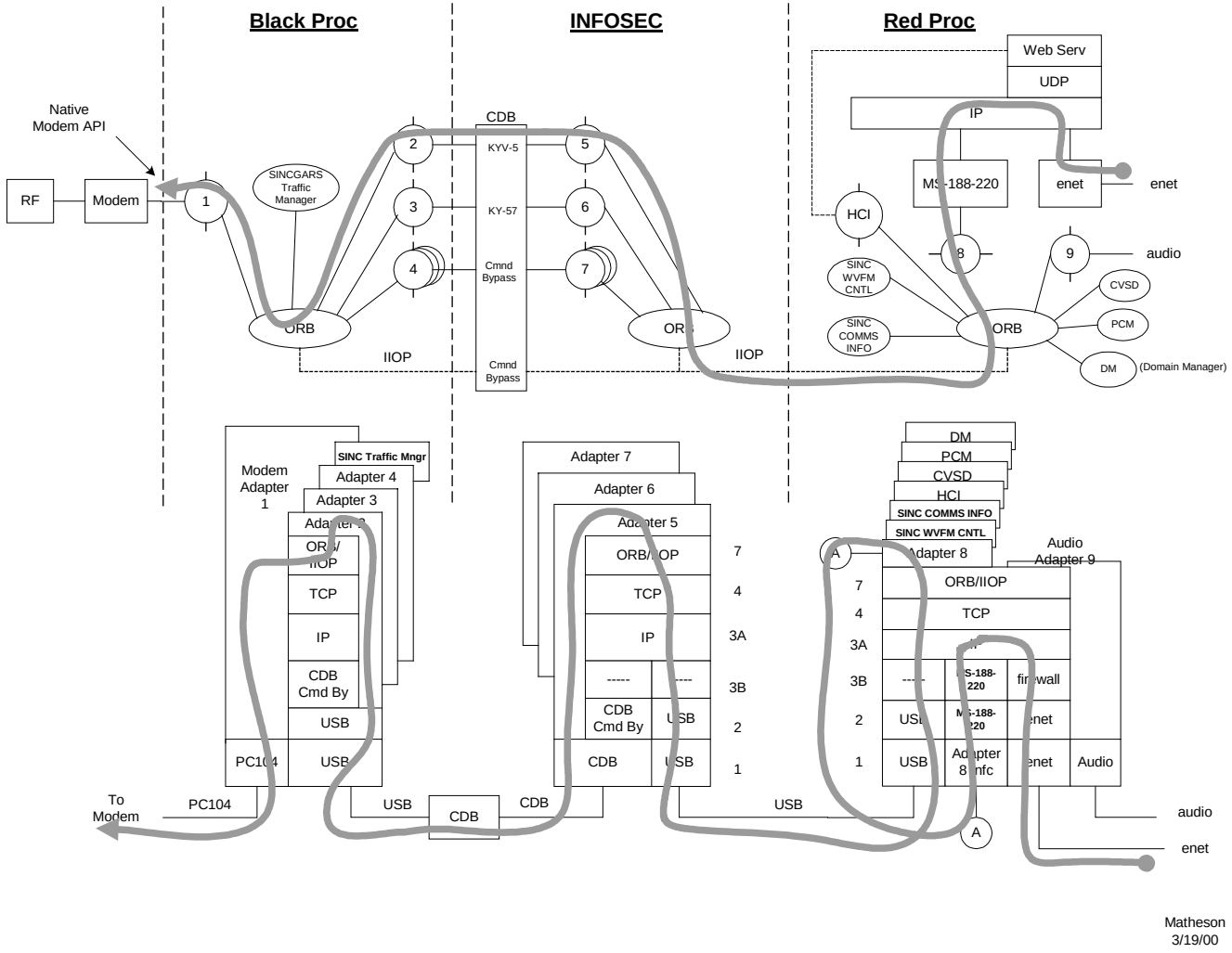


Figure 3: Waveform Data Traffic uses non-CORBA Transfer Mechanism across Red-Black Boundary

SINCGARS Packet and Voice on JTRS 2A Platform - Packet enters at TCP socket and output on Modem

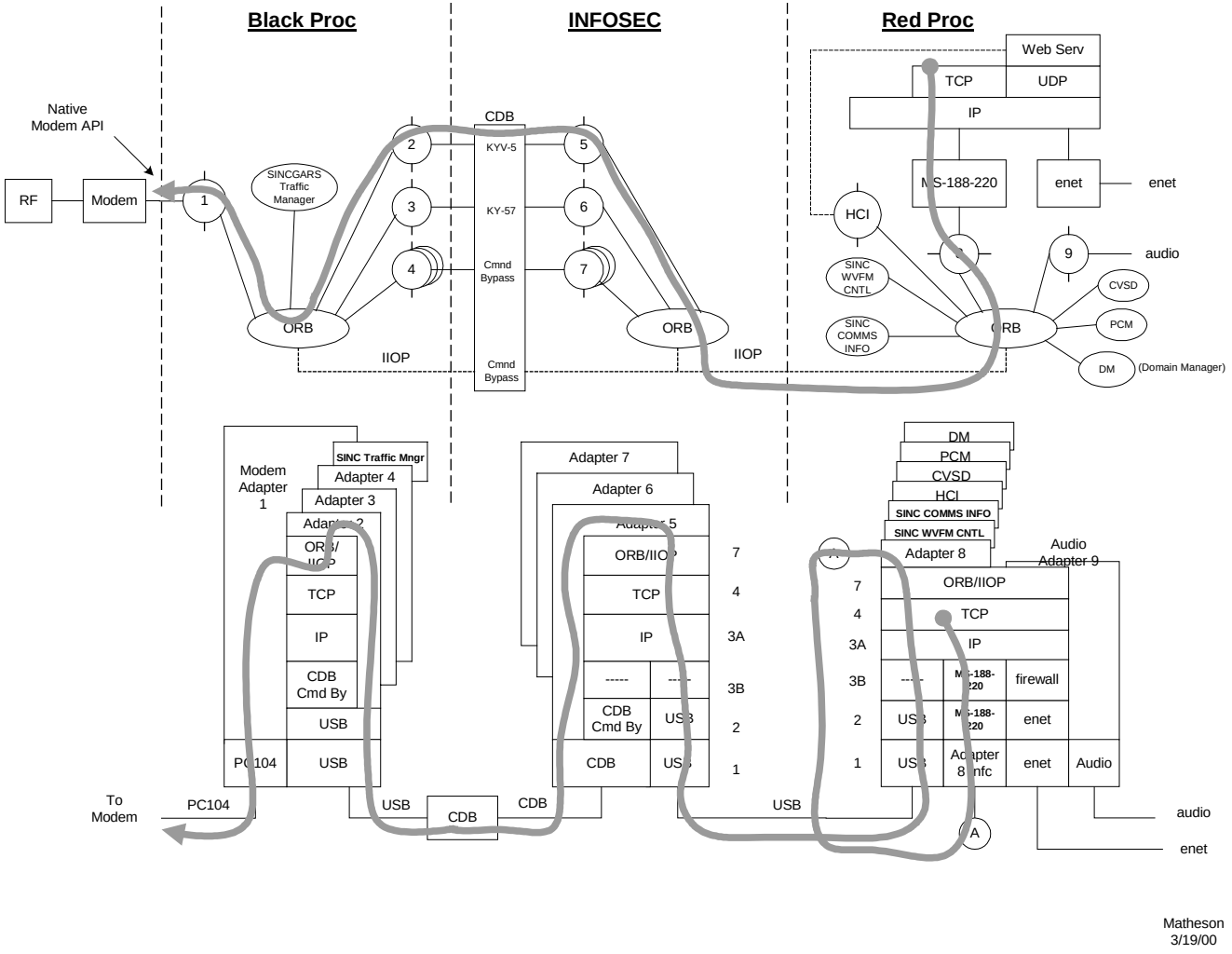


Figure 4: Waveform Data Traffic uses non-CORBA Transfer Mechanism across Red-Black Boundary

SINCGARS Packet and Voice on JTRS 2A Platform - Waveform Control (Bidirectional)

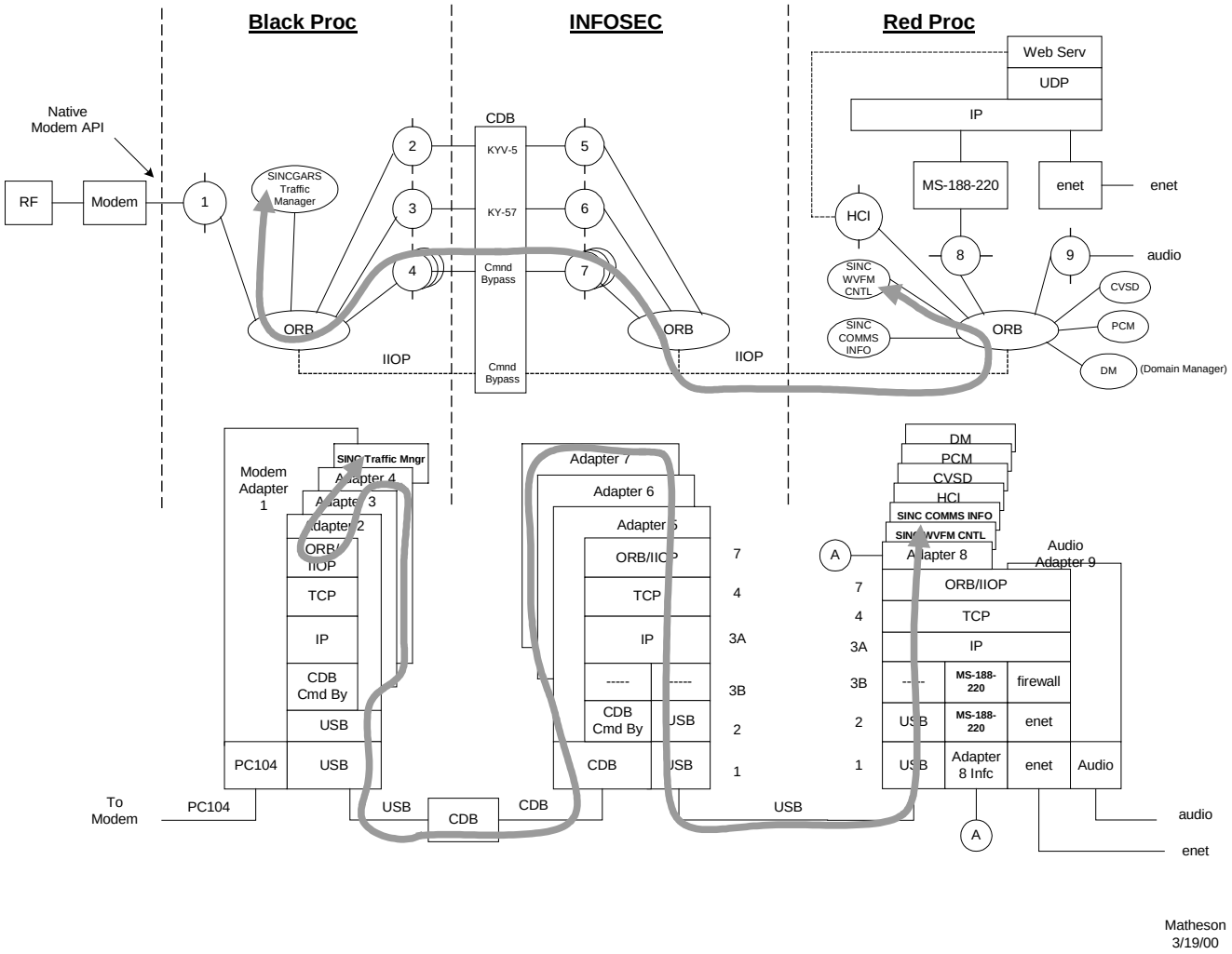


Figure 5: Waveform Control uses non_CORBA Transfer Mechanism across Red-Black Boundary

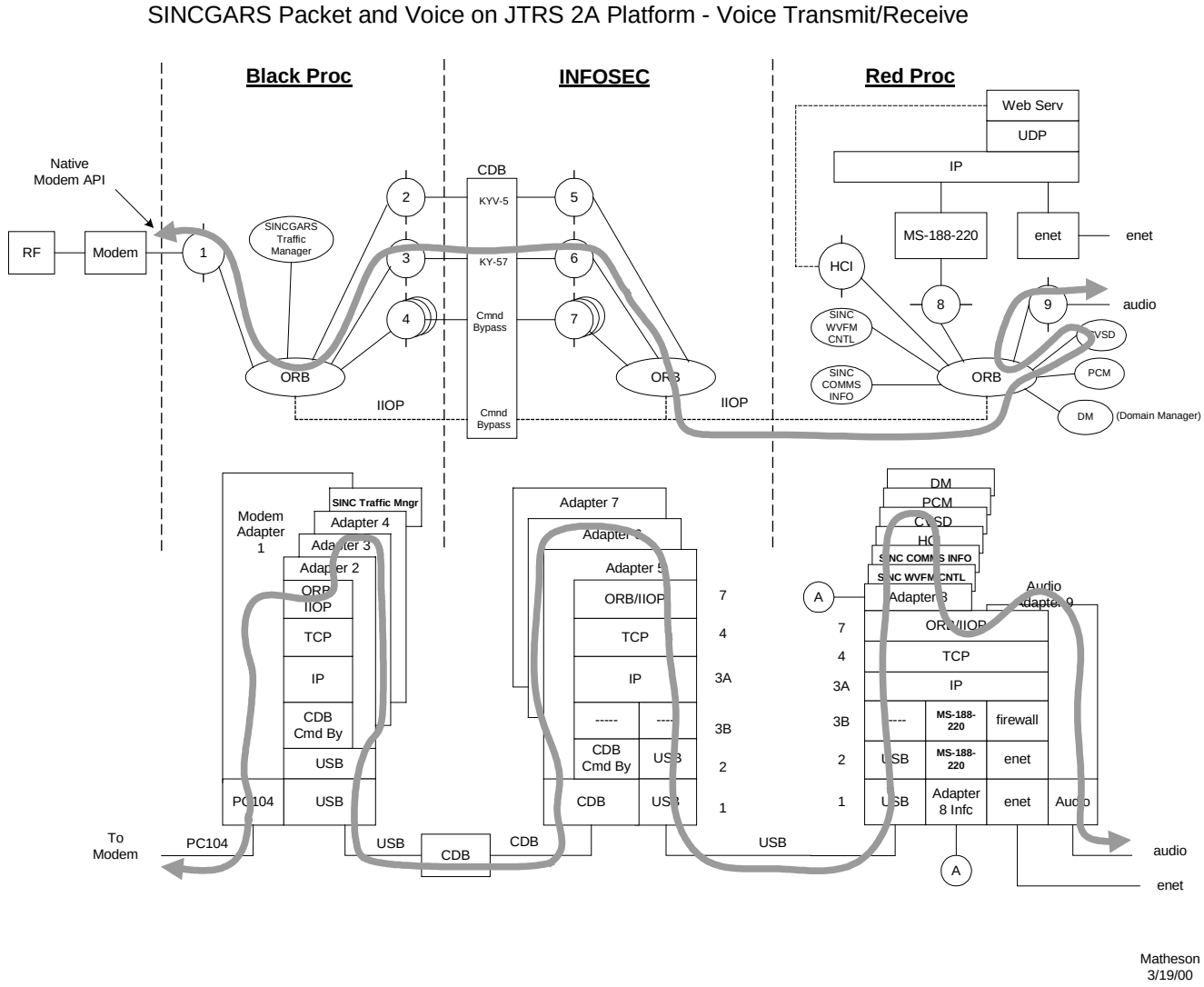


Figure 6: Waveform Voice Traffic uses non-CORBA Transfer Mechanism across Red-Black Boundary

3.2.2.4 OE

This approach uses the transfer mechanism services provided by the OE for communications between waveform objects in all cases except for transport across the Red-to-Black Boundary and within COTS and Legacy code objects.

3.2.2.5 Layer Applicability

Modem, Link, Network (3A, 3B), Transport

3.2.2.6 FOM Analysis

3.2.2.6.1 Waveform Portability

3.2.2.6.1.1 Porting Activity Identification

This section identifies the porting steps required for a given transfer mechanism alternative to be ported given certain stimuli.

3.2.2.6.1.1.1 Porting Stimuli Definitions

Stimulus	Description	Classification
Environment Change:Different Processor	Waveform must be ported to a different processor	Non-recurring
Environment Change:Different OS	Waveform must be moved to a different OS	Non-recurring
Environment Change:Different Bus	Waveform must be moved to a different bus structure	Non-recurring
Protocol Standard Migration:COTS Stack	Waveform requires new capabilities as reflected in new protocol standards (e.g. implementation of IP RFCs). The networking implementation uses COTS stack components.	Non-recurring
Protocol Standard Migration:Custom Stack	Waveform requires new capabilities as reflected in new protocol standards (e.g. implementation of IP RFCs). The networking implementation uses custom stack components.	Non-recurring
New Transport Protocol: COTS Stack	A new transport protocol must be added (e.g. to support reliable multicast). The networking implementation uses COTS stack components.	Non-recurring
New Transport Protocol: Custom Stack	A new transport protocol must be added (e.g. to support reliable multicast). The networking implementation uses custom stack components	Non-recurring

3.2.2.6.1.1.2 Porting Step Definitions

Porting Step	Description	Effort Level
Recompilation	Recompilation of networking part of waveform.	Minimal
Thin Layer Port	Port of OS/Kernel isolation layer	Moderate
Bus Layer Port	Port of bus communication layer	Moderate
Mechanism Port	Port of networking functionality to the chosen implementation mechanism (i.e. CORBA, STREAMS. Etc)	Extensive
Application Port	Implementation in a particular environment requires porting of one or more applications. For example a custom socket implementation may invalidate the use of select() in applications.	Minimal-Extensive
Capability Extension	Requires the extension of existing software to add new capabilities	Moderate-Extensive
OS Rebuild	Requires rebuilding the OS.	Minimal

3.2.2.6.1.1.3 Porting Steps Required for a Given Stimulus

Environment Change (non-recurring)			Waveform Protocol Standard Migration (non-recurring)		New Waveform Transport Protocol (non-recurring)	
Different Processor	**Different OS**	**Different Bus**	**COTS Stack**	**Custom Stack**	**COTS Stack**	**Custom Stack**
Recompilation	*Recompilation[1], Mechanism Port[2]*	*No Impact*	*OS rebuild.*	*Capability extension with OS rebuild.*	*OS rebuild.*	*Capability extension with OS rebuild.*

1. Dependent on an ORB being provided to abstract the OS.

2. If the OS stack implementations are different, i.e. BSD and STREAMS.

3.2.2.6.1.2 Impacts to CF caused by using the transfer mechanism.

None to minimal because the transfer mechanisms lie either below the CF (at the same or lower level than the GIOP) or above the CF.

3.2.2.6.2 Ability to support networking/non-networking waveforms

3.2.2.6.2.1 Supporting existing networking waveforms.

See Diagrams 3-1 to 3-4 for an example how SINCGARS Packet/INC is supported.

Use of adapters for COTS code (e.g. IP router, Web Server)

Use of adapters for Legacy Code (e.g. MS-188-220)

Use of adapters for non-CORBA capable hardware (e.g. modem adapter)

Use of CORBA transfer mechanism for everything except transport of Waveform real-time control and data traffic across Red-to-Black Boundary.

Use of Non-CORBA transfer mechanism across Red-to-Black Boundary for Waveform real-time control and data.

3.2.2.6.2.2 Supporting existing non-networking waveforms.

See Diagram 3-5 for an example how SINCGARS Voice is supported.

Use of adapters for non-CORBA capable hardware (e.g. audio and modem adapters)

Use of CORBA transfer mechanism for everything except transport of Waveform real-time control and data traffic across Red-to-Black Boundary.

Use of non-CORBA transfer mechanism across Red-to-Black Boundary for Waveform real-time control and data.

3.2.2.6.2.3 Supporting future networking waveforms.

Approach would still be to use CORBA transfer mechanism for everything except transport of Waveform real-time control and data traffic across Red-to-Black Boundary.

Overhead and performance of complete OE is a factor in support of high-throughput/low-latency future waveforms.

3.2.2.6.2.4 Supporting future non-networking waveforms.

Approach would still be to use CORBA transfer mechanism for everything except transport of Waveform real-time control and data traffic across Red-to-Black Boundary.

Overhead and performance of complete OE is a factor in support of high-throughput/low-latency future waveforms.

3.2.2.6.3 Performance

The following tables quantify in relative terms the steps necessary for a packet to traverse the radio infrastructure. Protocol specific processing and routing are not included as these steps would be common to all stack implementations.

3.2.2.6.3.1 Transfer Mechanism Packet Flow

3.2.2.6.3.1.1 Unicast Packet from Ethernet to Modem

Layers and Layer Transitions	Quantums
Ethernet Link Layer	allocate packet buffer + strip ethernet header
Transition to IP Network Layer	
IP Network Layer	IP Forwarding
Transition to MS-188-220 Module	Allocate packet buffer + copy packet + free IP packet buffer + context switch
MS-188-220 Module (Intranet and Link Layers)	Prepend Intranet and Link Headers
Transition to Adapter 8	Allocate packet buffer + copy packet + free MS-188-220 packet buffer + context switch
Transition to Red INFOSEC Adapter 5	Allocate packet buffer + copy packet + free Adapter 8 packet buffer + context switch + CORBA Transfer Mechanism overhead
Red INFOSEC Adapter	Copy packet into CRYPTO Engine + free packet buffer + context switch
CRYPTO Engine	Encrypt
Black INFOSEC Adapter	Allocate packet buffer + Copy packet from CRYPTO Engine + context switch
Transition to Modem Adapter (1)	Allocate packet buffer + copy packet + free Adapter 5 packet buffer + context switch + CORBA Transfer Mechanism overhead
Modem Adapter	Copy packet into Modem Dual Port Memory + free packet buffer + context switch
Modem Module	Copy packet from Modem Dual Port Memory

3.2.2.6.3.1.2 Multicast Packet from Ethernet to Modem(s) (3 channels)

Layers and Layer Transitions	Quantums
Ethernet Link Layer	allocate packet buffer + strip ethernet header
Transition to IP Network Layer	
IP Network Layer	IP Forwarding
Transition to MS-188-220 Module	(Allocate packet buffer + copy packet + free IP packet buffer + context switch) *3
MS-188-220 Module (Intranet and Link Layers)	(Prepend Intranet and Link Headers)*3
Transition to Adapter 8	(Allocate packet buffer + copy packet + free MS-188-220 packet buffer + context switch)*3
Transition to Red INFOSEC Adapter 5	(Allocate packet buffer + copy packet + free Adapter 8 packet buffer + context switch + CORBA Transfer Mechanism overhead)*3
Red INFOSEC Adapter	(Copy packet into CRYPTO Engine + free packet buffer + context switch)*3
CRYPTO Engine	Encrypt (all three will process in parallel)
Black INFOSEC Adapter	(Allocate packet buffer + Copy packet from CRYPTO Engine + context switch)*3
Transition to Modem Adapter (1)	(Allocate packet buffer + copy packet + free Adapter 5 packet buffer + context switch + CORBA Transfer Mechanism overhead)*3
Modem Adapter	(Copy packet into Modem Dual Port Memory + free packet buffer + context switch)*3
Modem Module	Copy packet from Modem Dual Port Memory (all three will process in parallel)

3.2.2.6.3.1.3 Message from Application to Modem using Sockets and TCP

Layers and Layer Transitions	**Quantums**
Transition from Application to Sockets Module	Allocate packet buffer(s) + copy packet + context switch
Socket Module	Translate Socket semantics to TPI semantics
Transition from Sockets Module to TCP Layer	
TCP Layer	Prepend TCP Header + Split Message + Increment Reference Count of buffer(s)
Transition from TCP Layer to IP Network Layer	
IP Network Layer	Prepend IP Header + IP Forwarding
Transition to MS-188-220 Module	Allocate packet buffer + copy packet + free IP packet buffer + context switch
MS-188-220 Module (Intranet and Link Layers)	Prepend Intranet and Link Headers
Transition to Adapter 8	Allocate packet buffer + copy packet + free MS-188-220 packet buffer + context switch
Transition to Red INFOSEC Adapter 5	Allocate packet buffer + copy packet + free Adapter 8 packet buffer + context switch + CORBA Transfer Mechanism overhead
Red INFOSEC Adapter	Copy packet into CRYPTO Engine + free packet buffer + context switch
CRYPTO Engine	Encrypt
Black INFOSEC Adapter	Allocate packet buffer + Copy packet from CRYPTO Engine + context switch
Transition to Modem Adapter (1)	Allocate packet buffer + copy packet + free Adapter 5 packet buffer + context switch + CORBA Transfer Mechanism overhead
Modem Adapter	Copy packet into Modem Dual Port Memory + free packet buffer + context switch
Modem Module	Copy packet from Modem Dual Port Memory

3.2.2.6.3.2 Identify possible QoS impacts

- *Possible Non-deterministic behavior*
- *Mapping into standard/evolving mechanisms*

3.2.2.6.4 Extensibility

3.2.2.6.4.1 Does the protocol implementation mechanism support addition of newly defined or extension to existing protocol layers?

Since this approach makes extensive use of CORBA transfer mechanism between application objects, addition of newly defined protocol layers is easy, and changes in waveform implementation (addition of waveform objects) is also easy.

Changes and extensions to messages passed via Non-CORBA transfer mechanism across the Red-to-Black boundary do require that the Bypass Adapters be modified.

3.2.2.6.5 Construct/Deconstruct Protocol Stacks

3.2.2.6.5.1 Address ability to add/remove layers without OS rebuild

Wherever CORBA transfer mechanism are used no OS rebuild is necessary to add/remove application layers. This is simply of function of waveform application instantiation/teardown. Modify the Domain Profile.

Where Non-CORBA transfer mechanism is used for transfer mechanism across the Red-to-Black boundary, no OS rebuild is necessary either. This is simply of function of waveform application instantiation/teardown.

Need support to add and remove protocol layers below the COTS IP stack. OS rebuild required for new transport layers, e.g. XTP.

3.2.2.6.5.2 Address ability to distribute protocol layers across address spaces.

This is fully supported.

3.2.2.6.5.3 Address dynamic construction/deconstruction of protocol stacks

Refer to x.6.5.1

3.2.2.6.6 Component Interoperability

3.2.2.6.6.1 Address how components can be used between waveforms

With respect to the transfer mechanism, use of CORBA transfer mechanism between waveform objects supports/enables component reuse. The limiting factors become component functionality and interface definition, not transfer mechanism.

3.2.2.6.6.2 Address how components interoperate across differing processors

Use of CORBA transfer mechanism makes this transparent. Non-CORBA transfer mechanism adapters running on different processors on the red and black sides must be concerned with data representation/interpretation.

3.2.2.6.7 Cost

Identify cost drivers and attempt to quantify in high level terms.

3.2.2.6.7.1 Implementation Cost

- Level of custom development – *Limited to waveform objects and bypass adapters only*
- Labor – *Use of CORBA transfer mechanism minimizes waveform development labor*
- Licenses - *None*
- COTS – *Use of CORBA transfer mechanism through adapters allows maximum use of COTS.*
- Integration – *Use of CORBA transfer mechanism allows integration to concentrate on application integration and not on infrastructure integration.*

3.2.2.6.7.2 Maintenance Cost

Effect of version migration of protocol standards (e.g. IP).

As protocol standards evolve waveform applications may have to be updated. This is relatively uncomplicated when CORBA transfer mechanism is used because effort is limited to application objects only.

Non-CORBA transfer mechanism adapters may also have to be updated for any new messages/formats.

3.2.2.6.7.3 Porting Cost

This is where use of CORBA transfer mechanism between application object is a clear winner. Use of CORBA transfer mechanism yields high portability between platforms and minimizes porting costs.

Non-CORBA transfer mechanism adapters are probably waveform specific so will have to be modified for any INFOSEC hardware differences between platforms.

3.2.2.6.8 Commercial Acceptance

3.2.2.6.8.1 Current Commercial Acceptance

SCA use of CORBA transfer mechanism for communications between distributed objects follows commercial precedent, so some degree of commercial acceptance can reasonably be expected. Rated as a 3.

- *No Commercial Acceptance*
- *Some Commercial Acceptance*
- *Widespread Commercial Acceptance*

Note: These can directly map to 1,3,9 rating system.

3.2.2.6.8.2 Future Commercial Acceptance

SCA use of CORBA transfer mechanism for communications between distributed objects follows commercial precedent, so some degree of commercial acceptance can reasonably be expected. Rated as a 3.

3.2.2.6.9 Security

3.2.2.6.9.1 Can support Local/Global Address Space?

3.2.2.6.9.2 Does transfer mechanism meet security requirements?

There is security concern about ability to monitor any messages transported by CORBA across the Red-to-Black boundary.

3.2.2.6.9.3 How does transfer mechanism handle control/status, header and plain text bypass?

Don't know, but possible security issue.

3.2.2.6.10 Ability to Adapt to Commercial Trends

Identify how the given transfer mechanism could support moving to other implementations.

Unknown.

3.2.3 Option 3 – CORBA, CORBA, CORBA via reference, CORBA via reference

Interface Type: Transfer Mechanism

Name: CORBA, CORBA, CORBA via reference, CORBA via reference

Date: 20 MAR 00

Revision: 5.0

Author: Fred Mabe

Company: Rockwell Collins (fdmabe@collins.rockwell.com)

3.2.3.1 Description

This alternative (option #3) uses CORBA for real/non-real time control and data flow. This option advocates when the there is a real time requirement which can not be met via the CORBA standard IOP, the standard CORBA IOP should be augmented. The augmentations discussed are: the ability to use CORBA pass by reference (e.g. shared memory transport and shared a RAM card) and/or grouping larger groups of protocol options together. Option #3 is a superset of transfer mechanism options #1 and #2 with the addition of a pass by reference when the real time requirements necessitate the speed.

3.2.3.2 Diagram

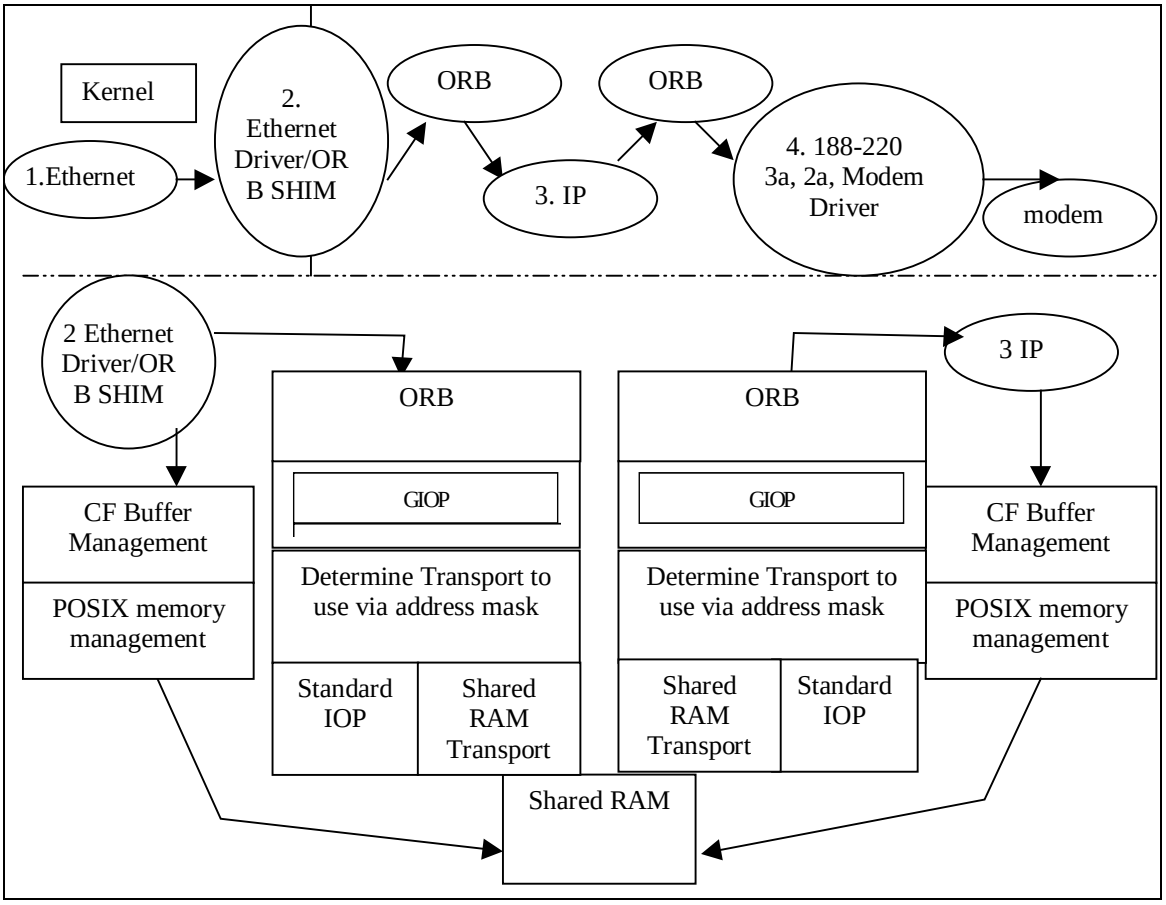


Figure 1

3.2.3.3 Theory of Operation

Figure 1 is an example of data flowing using transfer mechanism option #3. The goal of option #3 is to abstract changes to the transfer mechanism via the CF interface and the CORBA ORB.

In Figure 1 the top portion of the figure displays the data flow inside the radio. The following steps describe an incoming Ethernet packet as it flows through the radio to be transmitted on a 188-220 subnet:

1. An interrupt has been processed by the kernel, which indicates the an Ethernet packet has arrived.
2. The Ethernet driver/ORB shim uses a buffer extracted from the CF buffer interface to copy the Ethernet data via the driver. The Ethernet driver/ORB makes a standard CORBA call to send the data to the IP layer.
3. The IP layer routes the packet to the 188-220 subnet and makes a standard CORBA call to send the data to the 188-220 subnet.
4. The 188-220 3a & 2a processes the IP packet. Assuming the modem driver does reside on the same card as the 188-220 link layer, a call will be made to the Modem Driver. The Modem Driver will process the buffer, send it to the modem, and release the buffer.

The lower portion of figure #1 shows the details of the transfer mechanism interface between the Ethernet Driver/ORB SHIM and the IP object.

3.2.3.3.1 Red Side Processing and Black Side Processing

The method of data flow will be the same for both the red and the black side. The black and red separation will be accomplish the same way as option #2: each transport endpoint which crosses the INFOSEC boundary will have a bypass adapter which will check message content . CORBA bypass adapters will be waveform specific; so will have to be modified for any INFOSEC hardware differences between platforms (note: security mechanisms may have to be augmented/adjusted according to the outcome of the IPT Security working group).

3.2.3.4 OE

CORBA, POSIX OS, and CF memory management, which is similar to STREAMS memory management model, abstracts the OE from the application/protocol.

3.2.3.5 Layer Applicability

CORBA is applicable to all layers (Modem, Link, Network (3A, 3B), Transport). CORBA, POSIX OS, and CF abstracts the application/protocol from the application/protocol.

3.2.3.6 FOM Analysis

3.2.3.6.1 Waveform Portability

3.2.3.6.1.1 Porting Activity Identification

This section identifies the porting steps required for a given transfer mechanism alternative to be ported given certain Porting Activity Identification stimuli.

3.2.3.6.1.1.1 Porting Stimuli Definitions

Stimulus	Description	Classification
Environment Change:Different Processor	Waveform must be ported to a different processor	Non-recurring
Environment Change:Different OS	Waveform must be moved to a different OS	Non-recurring
Environment Change:Different Bus	Waveform must be moved to a different bus structure	Non-recurring
Protocol Standard Migration:COTS Stack	Waveform requires new capabilities as reflected in new protocol standards (e.g. implementation of IP RFCs). The networking implementation uses COTS stack components.	Non-recurring
Protocol Standard Migration:Custom Stack	Waveform requires new capabilities as reflected in new protocol standards (e.g. implementation of IP RFCs). The networking implementation uses custom stack components.	Non-recurring
New Transport Protocol: COTS Stack	A new transport protocol must be added (e.g. to support reliable multicast). The networking implementation uses COTS stack components.	Non-recurring
New Transport Protocol: Custom Stack	A new transport protocol must be added (e.g. to support reliable multicast). The networking implementation uses custom stack components	Non-recurring

3.2.3.6.1.1.2 Porting Step Definitions

Porting Step	Description	Effort Level
Recompilation	Recompilation of networking part of waveform.	Minimal
Thin Layer Port	Port of OS/Kernel isolation layer	Moderate
Bus Layer Port	Port of bus communication layer	Moderate
Mechanism Port	Port of networking functionality to the chosen implementation mechanism (i.e. CORBA, STREAMS. Etc)	Extensive
Application Port	Implementation in a particular environment requires porting of one or more applications. For example a custom socket implementation may invalidate the use of select() in applications.	Minimal-Extensive
Capability Extension	Requires the extension of existing software to add new capabilities	Moderate-Extensive
OS Rebuild	Requires rebuilding the OS.	Minimal

3.2.3.6.1.1.3 Porting Steps Required for a Given Stimulus

Environment Change (non-recurring)			Protocol Standard Migration (recurring)		New Transfer mechanism Protocol (recurring)	
Different Processor	**Different OS**	**Different Bus**	**COTS Stack**	**Custom Stack**	**COTS Stack**	**Custom Stack**
Recompilation	***No impact***[1, 3] *or* ***Recompilation***[1, 3] ***,***	***No impact***[1, 3, and 4] *(Bus abstracted by ORB) to code being ported. However ORB will have to be modified and/or recompiled.* ***Bus Layer Port .***	***Recompilation***[2] *or* ***Capability Extension***[1, 2,3 and 4]***:*** .	***Extend capability***[1, 3 and 4]***:***	***Recompilation***[2] *or* ***Capability Extension***[4]	***Recompilation***[2] *or* ***Capability Extension***[4]

1. Assuming SW developed for POSIX OS or is OS independent.

2. If capability extension is available commercially

3. Assuming ORB/CF and /Networking-API supports pass by reference interface: either by coping or actually passing by reference between processes.

4. If the capability requires extension of Networking-API, develop IDL for Networking-API to interface with ORB.

3.2.3.6.1.2 Assess Impacts to CF caused by using the transfer mechanism.

Add a class for buffer management to CF or to all Networking-APIs.

3.2.3.6.2 Ability to support networking/non-networking waveforms

Yes, this method will support networking/non-networking waveforms.

The design/decision process that should be followed:

1. Separate waveform into objects, which may be reusable.

2. Determine the real time constraints between objects.

3. Determine how to meet the real time requirements (CORBA by value, CORBA by reference, and/or coupling objects because of real time constraints).

See Figure 1 for an example.

3.2.3.6.2.1 Supporting existing networking waveforms

See Diagram in Figure 1 for an example how 188-220 is supported.

Use of adapters for COTS code (e.g. IP router, Web Server)

Use of adapters for Legacy Code (e.g. MS-188-220)

Use of adapters for non-CORBA capable hardware (e.g. modem adapter)

3.2.3.6.2.2 Supporting existing non-networking waveforms

Use of adapters for non-CORBA capable hardware (e.g. audio and modem adapters)

Use of CORBA transfer mechanism for everything except transport of Waveform real-time control and data traffic across Red-to-Black Boundary.

3.2.3.6.2.3 Supporting future networking waveforms.

Use of adapters for COTS code (e.g. IP router, Web Server)

Create new networking waveforms with a CORBA/IDL interface.

3.2.3.6.2.4 Supporting future non-networking waveforms.

Use of adapters for COTS code (e.g. IP router, Web Server)

Create new networking waveforms with a CORBA/IDL interface.

3.2.3.6.3 Performance

The following tables quantify in relative terms the steps necessary for a packet to traverse the protocol stack for a given stack implementation. Protocol specific processing and routing are not included as these steps would be common to all stack implementations. Transfer Mechanism Packet Flow

3.2.3.6.3.1 Transfer Mechanism Packet Flow

3.2.3.6.3.1.1 Unicast Packet from Ethernet to Modem

Layers and Layer Transitions	**Quantums**
Ethernet Link Layer (same or different processor)	• *Allocate packet buffer from CF/shared ram + strip Ethernet header* • *Insert ORB GIOP (does marshalling /security) :* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
IP Network Layer (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Insert ORB GIOP* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
hLink Module (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Prepend Link headers (note may use same schema as streams to have header buffer and body buffers separate.)* • *Insert ORB GIOP* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
Modem Module (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Free buffer* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*

3.2.3.6.3.1.2 Multicast Packet from Ethernet to Modem (3 channels)

Layers and Layer Transitions	**Quantums**
Ethernet Link Layer (same or different processor)	• *Allocate packet buffer from CF/shared ram + strip ethernet header* • *Insert ORB GIOP (does marshalling /security) :* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
IP Network Layer (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Increment reference count on packet for 3 channels (no copies assuming on same shared ram card)* • *Insert ORB GIOP* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
Link Module (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Prepend Link headers (note may use same schema as streams to have header buffer and body buffers separate.)* • *Insert ORB GIOP* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
Modem Module (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Free buffers * 3* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*

3.2.3.6.3.1.3 Message from Application to Modem using Sockets and TCP

Layers and Layer Transitions	Quantums
Application (same or different processor)	• *Allocate packet buffer from CF/shared ram insert application data* • *Insert ORB GIOP (does marshalling /security) :* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
TCP Module via sockets Networking-API (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Prepend TCP header (note may use same schema as streams to have header buffer and body buffers separate.)* • *Insert ORB GIOP* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
IP Network Layer (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Prepend IP header (note may use same schema as streams to have header buffer and body buffers separate.)* • *Insert ORB GIOP* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
Link Module (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Prepend Link header (note may use same schema as streams to have header buffer and body buffers separate.)* • *Insert ORB GIOP* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*
Modem Module (same or different processor)	• *Context Switch (same processor)* • *Remove ORB GIOP* • *Free buffers* *Note: May be optimized if in same processor as N-1 protocol or N+1 protocol; or not required if coupled as one CORBA object.*

3.2.3.6.3.2 Identify other possible QoS impacts

The QoS available in STREAMS is not currently addressed in this option. It is TBD how to QoS between CORBA objects.

3.2.3.6.4 Extensibility

3.2.3.6.4.1 Does the protocol transfer mechanism support addition of newly defined or extension to existing protocol layers?

Yes

3.2.3.6.5 Construct/Deconstruct Protocol Stacks

3.2.3.6.5.1 Address ability to add/remove layers without OS rebuild

- Application *(Yes)*
- Transport *(Yes)*
- Network *(Yes)*

- Link *(Yes)*
- Modem *(Yes)*

3.2.3.6.5.2 Address ability to distribute protocol layers across address spaces

Since all dependencies are on the ORB, which abstracts the transfer mechanism, applications or protocol layers can be moved transparently: it will only require a recompile when different processors are involved.

3.2.3.6.5.3 Dynamic stack construction/destruction

This question is addressed in 6.5.1. Stack construction/destruction can occur while other waveforms are running.

3.2.3.6.6 Component Interoperability

3.2.3.6.6.1 Address how components can be used between waveforms

Accomplish via CORBA standard mechanism.

3.2.3.6.6.2 Address how components interoperate across differing processors

Accomplish via CORBA standard mechanism.

3.2.3.6.6.3 Address dynamic construction/deconstruction of protocol stacks.

Accomplish via CORBA standard mechanism.

3.2.3.6.7 Cost

Identify cost drivers and attempt to quantify in high level terms.

3.2.3.6.7.1 Implementation Cost

- Level of custom development – *Limited to waveform objects and bypass adapters only.*
- Labor – *Use of CORBA middleware minimizes waveform development labor.*
- Licenses – *The ORB.*
- COTS – *Use of CORBA transfer mechanism through adapters allows maximum use of COTS.*
- Integration – *Use of CORBA transfer mechanism allows integration to concentrate on application integration and not on infrastructure integration.*

3.2.3.6.7.2 Maintenance Cost

As protocol standards evolve waveform applications may have to be updated. This is relatively uncomplicated when the CORBA middleware is used because effort is limited to the object itself; CORBA & CF abstract the environment.

CORBA Bypass adapters may also have to be updated for any new messages/formats.

3.2.3.6.7.3 Porting Cost

This is where the use of CORBA transfer mechanism between application object is a clear winner. Use of CORBA transfer mechanism yields high portability between platforms and minimizes porting costs.

CORBA Bypass adapters are probably waveform specific so will have to be modified for any INFOSEC hardware differences between platforms.

3.2.3.6.8 Commercial Acceptance

3.2.3.6.8.1 Current Commercial Acceptance

SCA use of CORBA as middleware for communications between distributed objects follows commercial precedent, so some degree of commercial acceptance can expected.

Note: These can directly map to 1,3,9 rating system.

3.2.3.6.8.2 Future Commercial Acceptance

This is essentially crystal ball stuff and should receive less weight than current commercial acceptance.

3.2.3.6.9 Security

3.2.3.6.9.1 Can support Local/Global Address Space

3.2.3.6.9.2 Does transfer mechanism meet security requirements?

Unknown

3.2.3.6.9.3 How does the transfer mechanism handle control/status, header and plain text bypass?

Unknown

3.2.3.6.10 Ability to Adapt to Commercial Trends

Identify how the given transfer mechanism could support moving to other implementations.

Unknown

3.2.4 Option 4 – CORBA, CORBA, STREAMS, STREAMS

Interface Type: Transfer Mechanism

Name: CORBA, CORBA, STREAMS, STREAMS

Date: 20 MAR 00

Revision: 3.0

Author: Bruce Baker

Company: Raytheon

3.2.4.1 Description

This alternative uses STREAMS for real time data flow and real time control for networking protocol layers. It also uses STREAMS drivers as the low level interface to the INFOSEC and for the low-level interface to the modem. The STREAMS alternative is meant be used in cases where performance and QoS issues dictate a more efficient transfer mechanism between protocol layers. In addition it attempts to make maximum use of COTS networking protocol implementations.

3.2.4.2 Diagram

Diagrams are provided for clarity throughout the sections of this paper. No one diagram will be sufficient for explanatory purposes.

3.2.4.3 Theory of Operation

STREAMS provides the ability to plug/unplug protocol modules that provide efficient message passing between modules, buffer management, and flow control. There are also existing standards based on STREAMS that define network layer interfaces for link, network, and transport. This allows the use of COTS networking components and allows the vendor to concentrate on radio system development. The following sections address the instantiation of a networking waveform that requires red side link processing in order to illustrate how STREAMS would fit into the architecture.

3.2.4.3.1 Red Side Processing

Figure 1 shows a red side STREAMS based protocol stack after startup including relevant object representations. The Domain Manager (DoM), Red Processor Device (RP Dev) and Security Device (Sec Dev) and Object Request Broker (ORB) are shown. CORBA sits at the application layer and interfaces to the stack through the sockets API. The routing daemon which contains the routing protocols such as RIP, OSPF, DVMRP, etc also interfaces to the stack through the sockets API. At the bottom of the stack are two drivers, one for Ethernet and one for the INFOSEC. The Ethernet driver presents the Data Link Provider Interface (DLPI) interface to the IP module. The INFOSEC driver implements the interface to the INFOSEC. The INFOSEC driver does not present the DLPI interface itself because a link module may be pushed on top of it that is sending to and receiving from a modem module. Therefore the driver must be more generic. A separate DLPI module presents the DLPI interface to the IP module. As shown on

the diagram, external IIOP traffic flows through the Ethernet and up the protocol stack to the ORB or gets routed by the IP layer to the INFOSEC interface for transport to a Black Processor. The IIOP traffic carries non-real time control and status information. Since there are no waveforms running, there is no real time data flow.

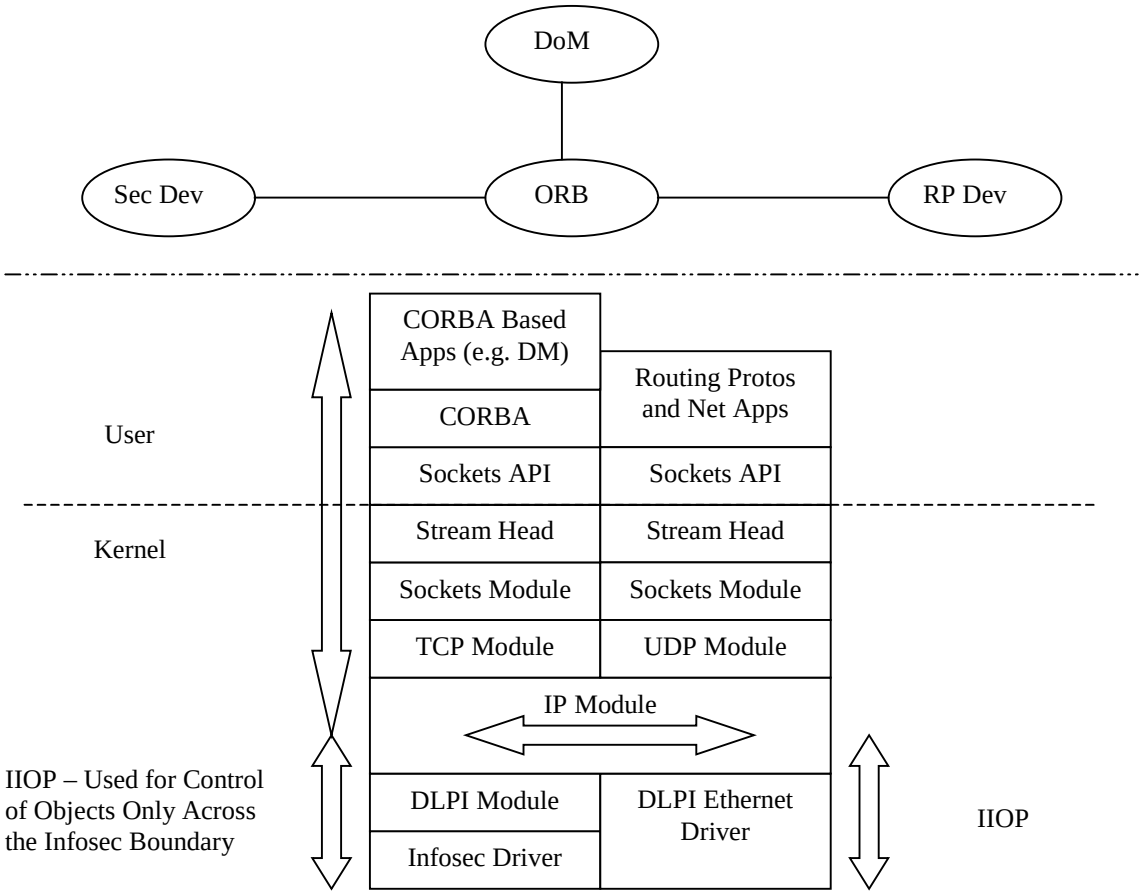


Figure 1- Red Side (Single Processor) Stack After Startup

Figure 2 shows the stack at the beginning of waveform instantiation. The RP Dev has been commanded by the Domain Manager to load (push) the link layer STREAMS module on top of the security device. RP Dev also imparts any persistent configuration information to the module at this point.

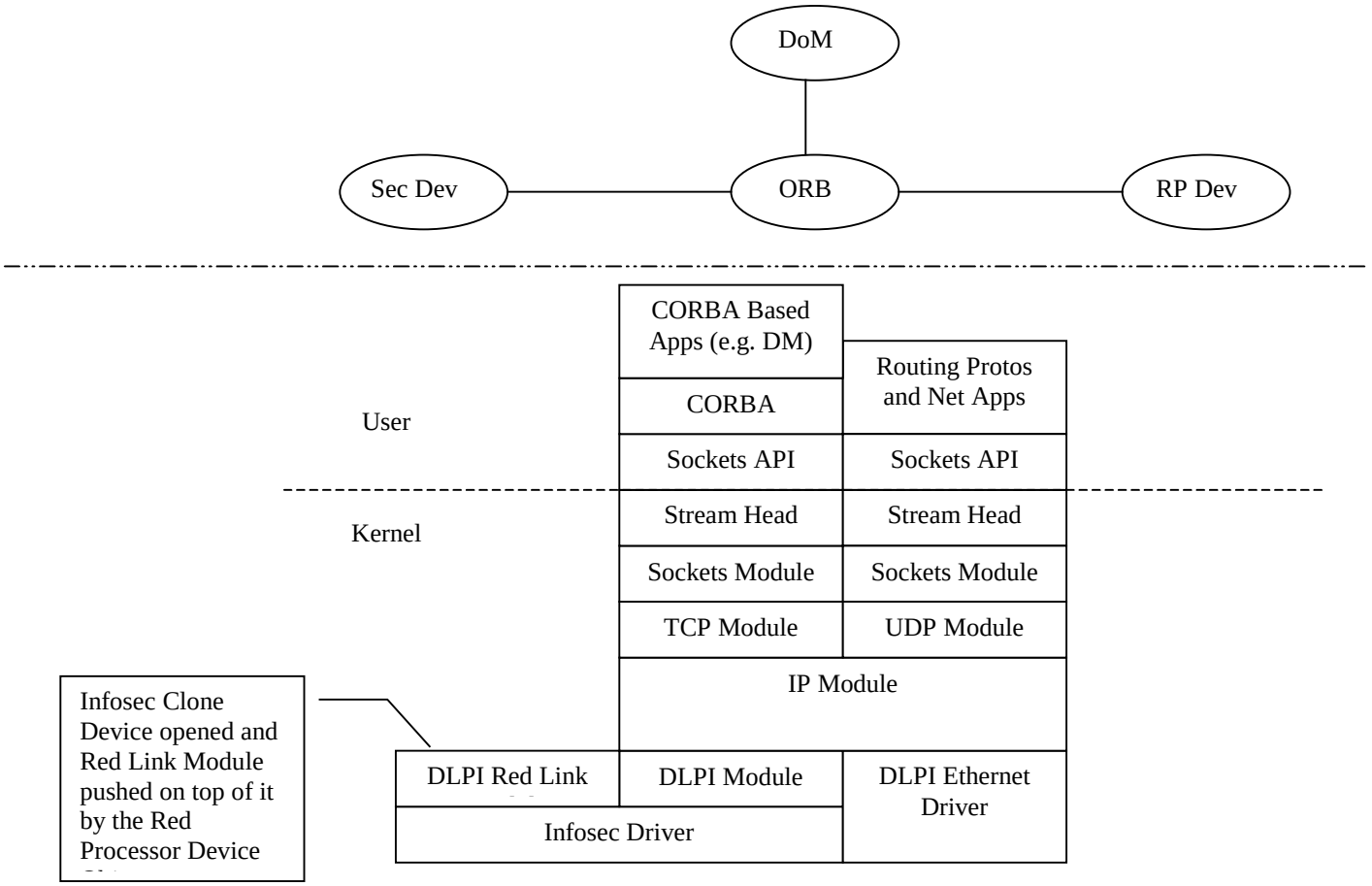


Figure 2 Red Side Stack Waveform Instantiation Step 1

Figure 3 shows the next step in the instantiation process. As part of the loading of the link layer STREAMS module, RP Dev commands the module to be persistently linked under the IP module. In addition an IP address is assigned to the interface. The interface can now be configured as an IP routing gateway. Assuming that the waveform has been fully instantiated and configured on the black side as well, the Radio System is ready to receive and transmit networked traffic over the air.

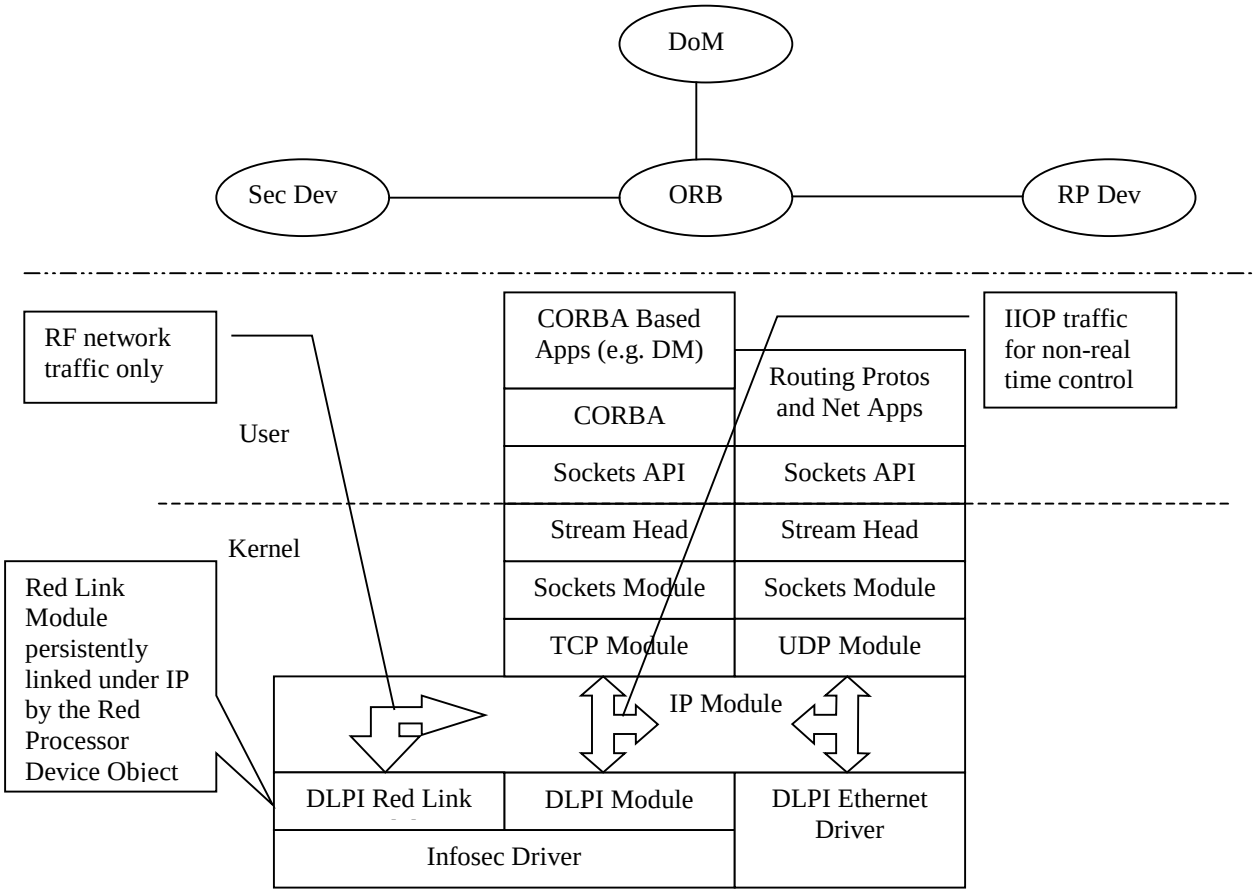


Figure 3 Red Side Stack Waveform Instantiation Step 2

3.2.4.3.2 Black Side Processing

Figure 4 shows a black side STREAMS based protocol stack after startup including relevant object representations. The Black Processor Device (BP Dev), Security Device (Sec Dev), Modem Device (Mod Dev), Modem Factory (Mod Fac) and ORB are shown. As on the red side CORBA sits at the application layer and interfaces to the stack through the sockets API. The routing daemon (optional) which contains the routing protocols such as RIP, OSPF, DVMRP, etc also interfaces to the stack through the sockets API. At the bottom of the stack are two drivers, one for the backplane and one for the INFOSEC. The INFOSEC and backplane drivers are separate from the DLPI modules. As shown on the diagram, IIOP traffic flows between this processor and other black processors and CORBA capable modems and can carry both non-realtime and real time traffic. IIOP traffic through the INFOSEC carries non-real time control and status information. Since there are no waveforms running, there is no real time data flow.

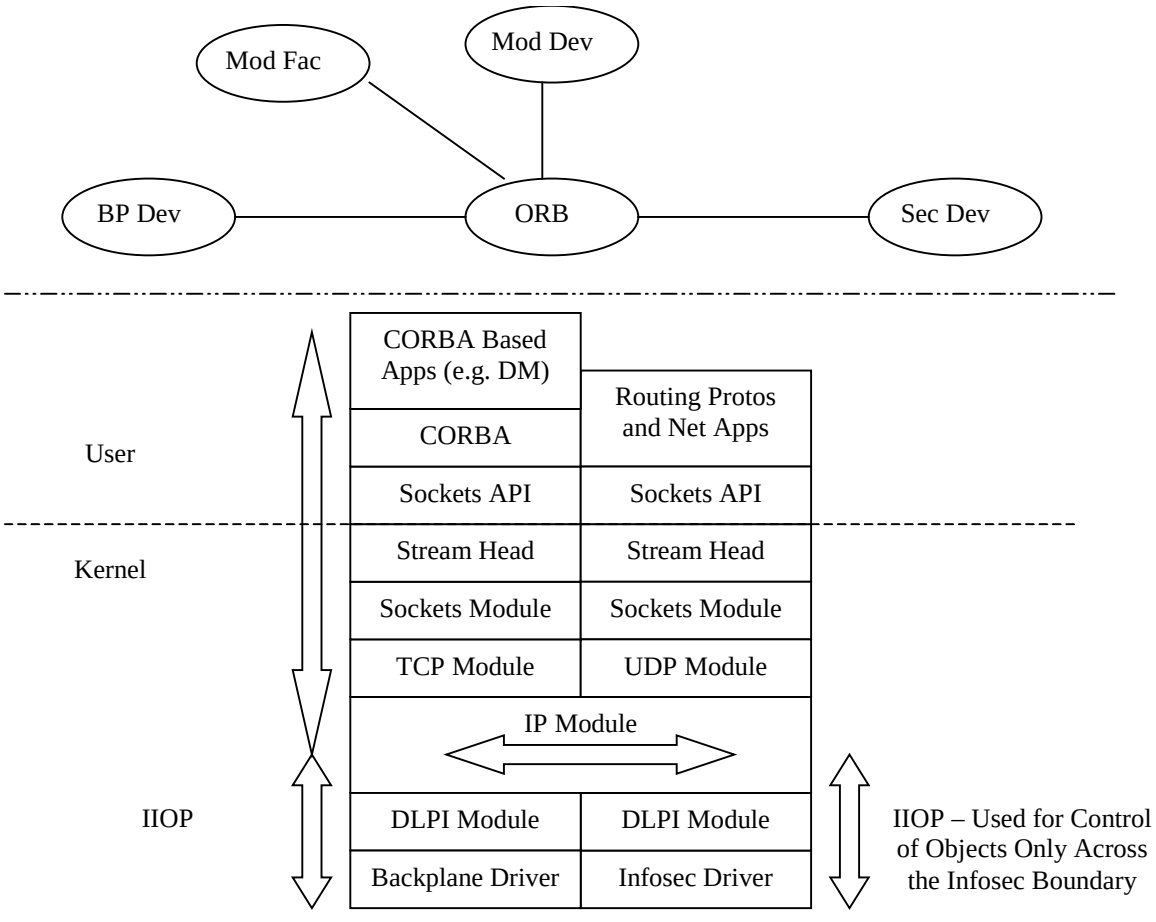


Figure 4 Black Side Stack After Startup

Figure 5 shows the stack at the beginning of waveform instantiation. The BP Dev has been commanded by the Domain Manager to load the STREAMS bridge multiplexor and link the

backplane and security device drivers under it. The BP Dev imparts persistent configuration information to the multiplexor at this point.

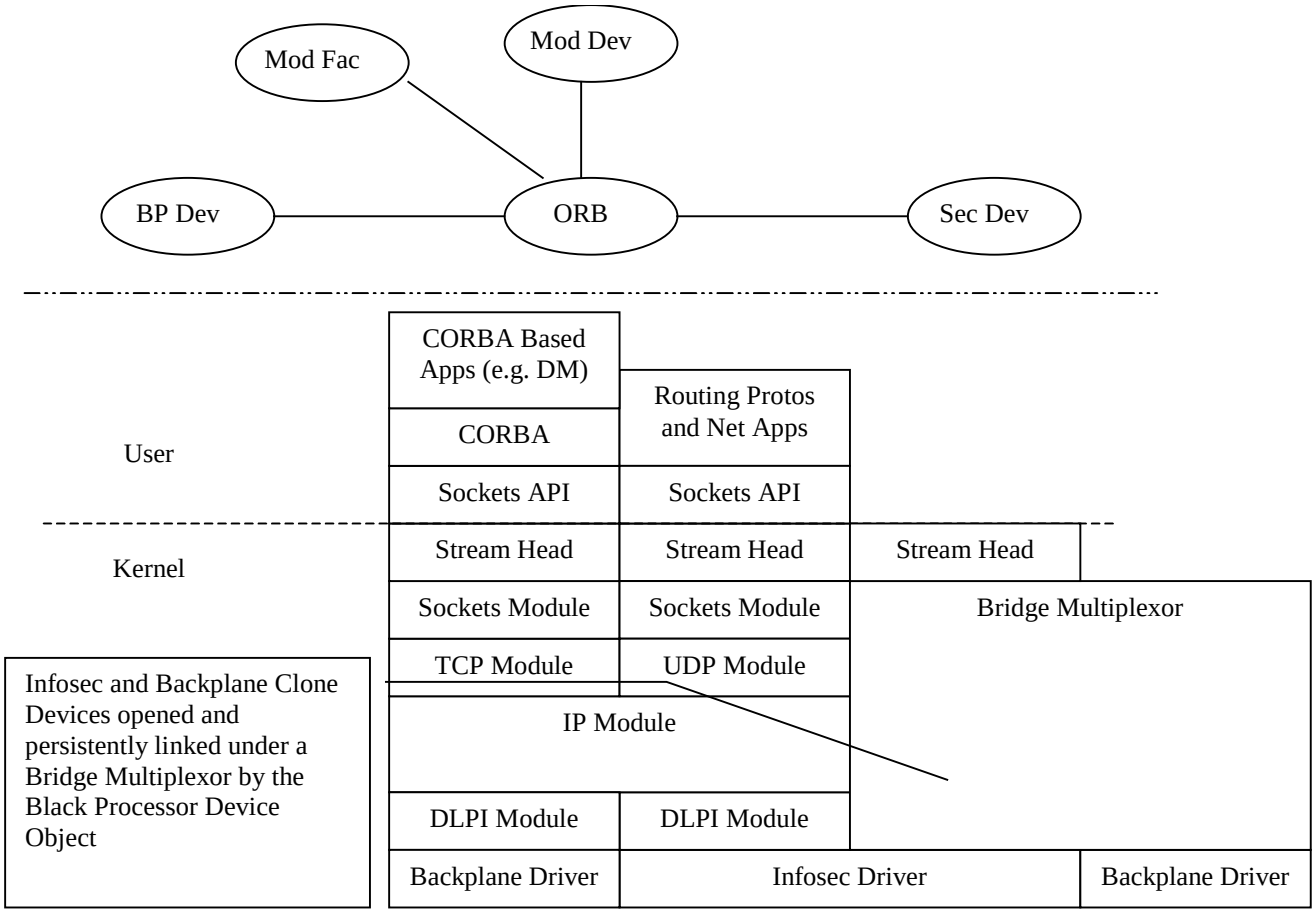


Figure 5 Black Side Stack Waveform Instantiation Step 1

Figure 6 shows the next step in the instantiation process. The Domain Manager commands the modem factory to create a modem adapter. The modem adapter exports a Modem Networking-API as its client interface. The server side of the adapter translates between the STREAMS API and the operations as defined in the Networking-API. This applies to non-real time control only. Real-Time (in-band) control and data flow through the bridge.

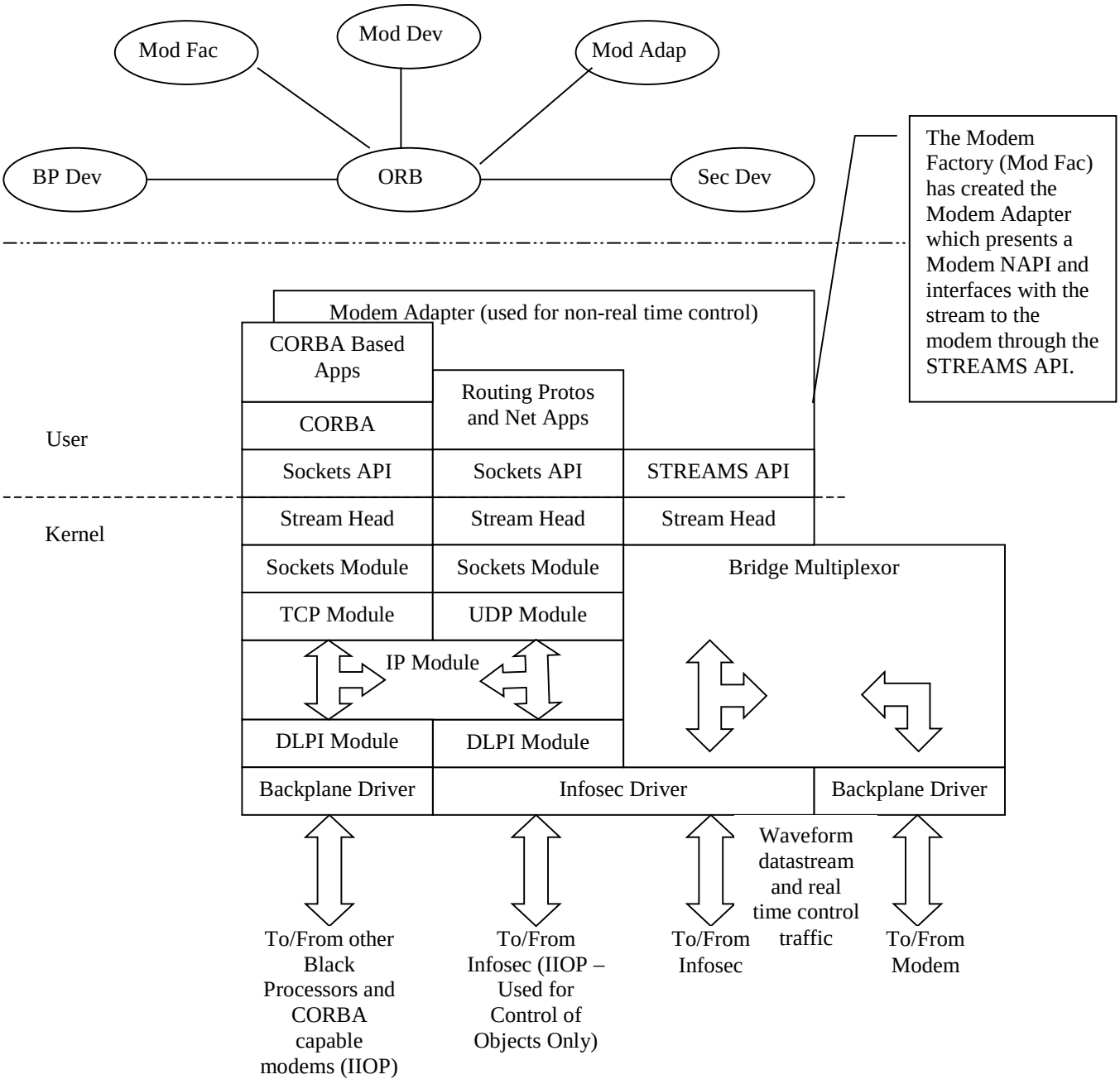


Figure 6 Black Side Stack Waveform Instantiation Step 2

3.2.4.4 OE

STREAMS based protocols depend on the existence of a STREAMS environment. This can be provided by an OS with a native STREAMS implementation or by a third party STREAMS environment. STREAMS is not restricted to any particular OS or OS type. In fact STREAMS can exist with no OS at all if you want to extend the STREAMS interface to the modem.

3.2.4.5 Layer Applicability

STREAMS is applicable to all layers (Modem, Link, Network (3A, 3B), Transport). In addition STREAMS has applicability for the Security interface.

3.2.4.6 FOM Analysis

3.2.4.6.1 Waveform Portability

3.2.4.6.1.1 Porting Activity Identification

This section identifies the porting steps required for a given transfer mechanism alternative to be ported given certain stimuli.

3.2.4.6.1.1.1 Porting Stimuli Definitions

Stimulus	**Description**	**Classification**
Environment Change:Different Processor	Waveform must be ported to a different processor	Non-recurring
Environment Change:Different OS	Waveform must be moved to a different OS	Non-recurring
Environment Change:Different Bus	Waveform must be moved to a different bus structure	Non-recurring
Protocol Standard Migration:COTS Stack	Waveform requires new capabilities as reflected in new protocol standards (e.g. implementation of IP RFCs). The networking implementation uses COTS stack components.	Non-recurring
Protocol Standard Migration:Custom Stack	Waveform requires new capabilities as reflected in new protocol standards (e.g. implementation of IP RFCs). The networking implementation uses custom stack components.	Non-recurring
New Transport Protocol: COTS Stack	A new transport protocol must be added (e.g. to support reliable multicast). The networking implementation uses COTS stack components.	Non-recurring
New Transport Protocol: Custom Stack	A new transport protocol must be added (e.g. to support reliable multicast). The networking implementation uses custom stack components	Non-recurring

3.2.4.6.1.1.2 Porting Step Definitions

Porting Step	Description	Effort Level
Recompilation	Recompilation of networking part of waveform.	Minimal
Thin Layer Port	Port of OS/Kernel isolation layer	Moderate
Bus Layer Port	Port of bus communication layer	Moderate
Mechanism Port	Port of networking functionality to the chosen implementation mechanism (i.e. CORBA, STREAMS. Etc)	Extensive
Application Port	Implementation in a particular environment requires porting of one or more applications. For example a custom socket implementation may invalidate the use of select() in applications.	Minimal-Extensive
Capability Extension	Requires the extension of existing software to add new capabilities	Moderate-Extensive
OS Rebuild	Requires rebuilding the OS.	Minimal

3.2.4.6.1.1.3 Porting Steps Required for a Given Stimulus

Environment Change (non-recurring)			Protocol Standard Migration (Non-recurring)		New Transport Protocol (Non-recurring)	
Different Processor	**Different OS**	**Different Bus**	**COTS Stack**	**Custom Stack**	**COTS Stack**	**Custom Stack**
Recompilation. This may require a translation module as discussed in 6.6.2.	*Thin Layer Port [1], OS Rebuild [1]*	*Bus Layer Port, OS Rebuild*	*Recompilation [2] or Capability Extension*	*Capability Extension*	*Recompilation [2] or Capability Extension*	*Recompilation [2] or Capability Extension*

1. For OSes which do not have either native or third party STREAMS support (approx 35 C functions).

2. If capability extension is available commercially.

3.2.4.6.1.2 Impacts to CF caused by using the transfer mechanism.

The impacts to the CF are minimal and amount to some additional profile definition. In particular to identify a module as a STREAMS module and identify any configuration information for the module that belongs in the profile.

3.2.4.6.2 Ability to support networking/Non-Networking Waveforms

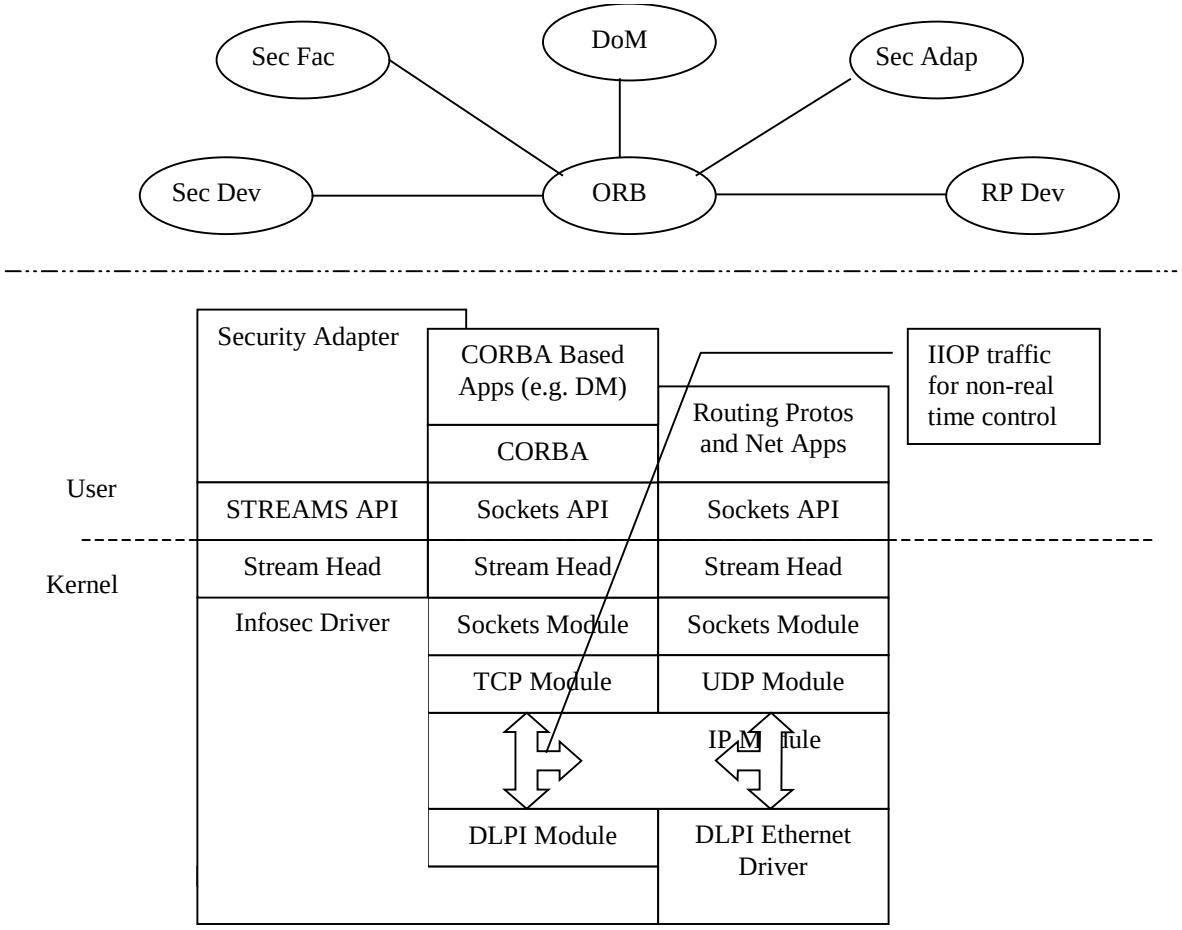
3.2.4.6.2.1 Supporting existing networking waveforms.

STREAMS is able to support legacy networking waveforms such as SINCGARS SIP and EPLRS. For SINCGARS SIP the layer 3b and 2 functionality of 188-220 would be encapsulated in a STREAMS module that would present DLPI as its interface. Figures 1-6 show how the networking portion of SINCGARS SIP would be instantiated. In the case of EPLRS the Black Side Link Layer would be a STREAMS module that also would present DLPI as its interface.

3.2.4.6.2.2 Supporting existing non-networking waveforms.

The figure below illustrates how STREAMS is integrated with legacy non-networking waveforms. The Domain Manager invokes the Security Factory. The Security Factory opens the clone

device for the security driver, creates a security adapter object and passes the file descriptor of the open stream to the adapter. A waveform such as FM LOS can now be run by creating an object that interfaces to an input device such as an audio device and linking the audio object with the security adapter.



3.2.4.6.2.3 Supporting future networking waveforms.

STREAMS can support future networking waveforms by the addition of STREAMS modules that support those waveforms.

3.2.4.6.2.4 Supporting future non-networking waveforms.

Future non-networking waveforms would be supported identically to legacy non-networking waveforms.

3.2.4.6.3 Performance

The following tables quantify in relative terms the steps necessary for a packet to traverse the protocol stack for a given stack implementation. Protocol specific processing and routing are not included as these steps would be common to all stack implementations. There has been concern that poor performance exhibited in certain STREAMS implementations is a result of inherent

STREAMS limitations. Investigation into this matter proves this not to be the case. The poor performance of some STREAMS implementations can be attributed to the implementations themselves and is not an inherent limitation of STREAMS.

3.2.4.6.3.1 Transfer Mechanism Packet Flow

3.2.4.6.3.1.1 Unicast Packet from Ethernet to Modem

Layers and Layer Transitions	**Quantums**
Ethernet Link Layer	*allocate packet buffer + strip ethernet header*
Transition to IP Network Layer	
IP Network Layer	
Transition to Link Module (same processor as network module)	
Transition to Link Module (different processor from network module)	*prepend inter-processor transport header + copy packet into transport buffer + free packet buffer + strip inter-processor transport header + allocate packet buffer + copy into packet buffer*
Link Module (DLPI)	*prepend link header*
Transition to Modem Module (same processor as link module)	
Transition to Modem Module (different processor from link module)	*prepend inter-processor transport header + copy packet into transport buffer + free packet buffer + strip inter-processor transport header + allocate packet buffer + copy into packet buffer*
Modem Module	*free packet buffer*

3.2.4.6.3.1.2 Multicast Packet from Ethernet to Modem (3 channels)

Layers and Layer Transitions	**Quantums**
Ethernet Link Layer	*allocate packet buffer + strip ethernet header*
Transition to IP Network Layer	
IP Network Layer	*Increment reference count on packet for 3 channels (no copies)*
Transition to Link Module (same processor as network module)	
Transition to Link Module (different processor from network module)	*(prepend inter-processor transport header + copy packet into transport buffer) * 3 + free packet buffer + (strip inter-processor transport header + allocate packet buffer + copy into packet buffer) * 3*
Link Module (DLPI)	*(allocate link header buffer + prepend link header) * 3*
Transition to Modem Module (same processor as link module)	
Transition to Modem Module (different processor from link module)	*(prepend inter-processor transport header + copy packet into transport buffer + free link header buffer) * 3 + free packet buffer + (strip inter-processor transport header + allocate packet buffer + copy into packet buffer) * 3*
Modem Module	*free packet buffer * 3*

3.2.4.6.3.1.3 Message from Application to Modem using Sockets and TCP

Layers and Layer Transitions	Quantums
Stream Head	*allocate buffer (s) + copy message to buffer(s)*
Transition to Sockets Module	
Sockets Module	*Translate socket semantics to TPI semantics*
Transition to TCP Module	
TCP Module (TPI)	*Split message Increment reference count of buffer chain + prepend TCP header*
Transition to IP Network Layer	
IP Network Layer	
Transition to Link Module (same processor as network module)	
Transition to Link Module (different processor from network module)	*prepend inter-processor transport header + copy packet into transport buffer + free packet buffer*
Link Module (same processor as network module)	*prepend link header*
Transition to Modem Module (same processor as link module)	
Transition to Modem Module (different processor from link module)	*prepend inter-processor transport header + copy packet into transport buffer + free packet buffer*
Modem Module	*free packet buffer*

3.2.4.6.3.2 Possible QoS impacts

The availability of priority banding in STREAMS may aid in implementing QoS policies. It does not appear to be in much use at this time.

3.2.4.6.4 Extensibility

3.2.4.6.4.1 Does the protocol transfer mechanism support addition of newly defined or extension to existing protocol layers?

STREAMS allows extension of protocols and definition of new protocols. Existing examples are Mentat's and Spider's products. Mentat has a TCP/IP stack that supports both IPv4 and IPv6. They continue to incorporate RFCs into the IPv6 module. In addition they supply an Xpress Transport Protocol module (XTP) that can be pushed on top of IP in place of TCP. Spider offers an SS7 stack and a Q3 Telecom stack.

3.2.4.6.5 Construct/Deconstruct Protocol Stacks

3.2.4.6.5.1 Ability to add/remove layers without OS rebuild

The ability to add remove the link and modem layers was shown in the theory of operation. Figure 8 shows an XTP module pushed on top of IP as an example of a transport layer that can be added without and OS rebuild.

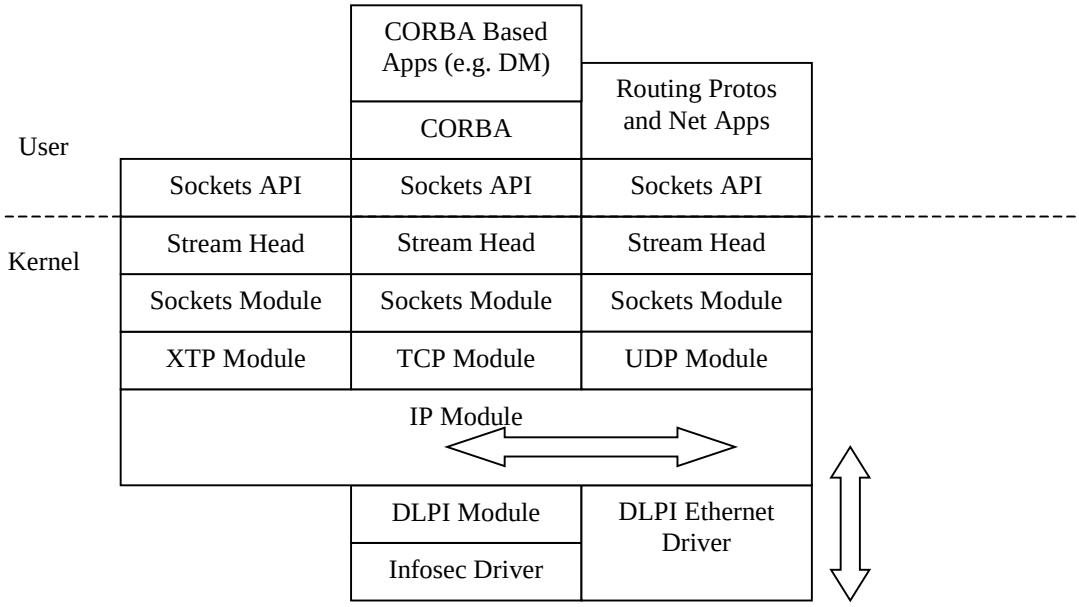


Figure 8 XTP Transport Pushed on Top of IP

The network layer (IP) could be replaced just as any other STREAMS module with the exception that the IP layer is used by multiple applications and interfaces that replacing it would entail shutting down all waveforms that use it.

Application addition and deletion are handled by the CF and is outside of STREAMS.

3.2.4.6.5.2 Address ability to distribute protocol layers across address spaces

This is essentially addressed in x.6.6.2.

3.2.4.6.5.3 Dynamic stack construction/destruction

This question is addressed in x.6.5.1. Stack construction/destruction can occur while other waveforms are running.

3.2.4.6.6 Component Interoperability

3.2.4.6.6.1 Address how components can be used between waveforms

The STREAMS paradigm is that the internals of any STREAMS module is unknown to modules above or below it in the stream. Only the interface of the module below is known. This allows the concept of different transport or link layers to run on top of or below IP for example. Based on the diagrams shown previously it can be seen that waveforms that have common elements such as a common link layer can be plugged in on top of or below waveform specific elements.

3.2.4.6.6.2 Address how components interoperate across differing processors

This alternative uses CORBA for non-realtime control. Interoperation across differing processors is handled by the CF interfaces and CORBA in this case except for the case were the devices are not CORBA capable. For networking STREAMS modules are used to handle the data STREAM. As shown in previous diagrams, STREAMS drivers are used to communicate with the INFOSEC. These drivers can ensure interoperability between themselves via the equivalent of marshalling. As shown in Figure 9 another method would be to push a translation module on top of the driver to perform this function.

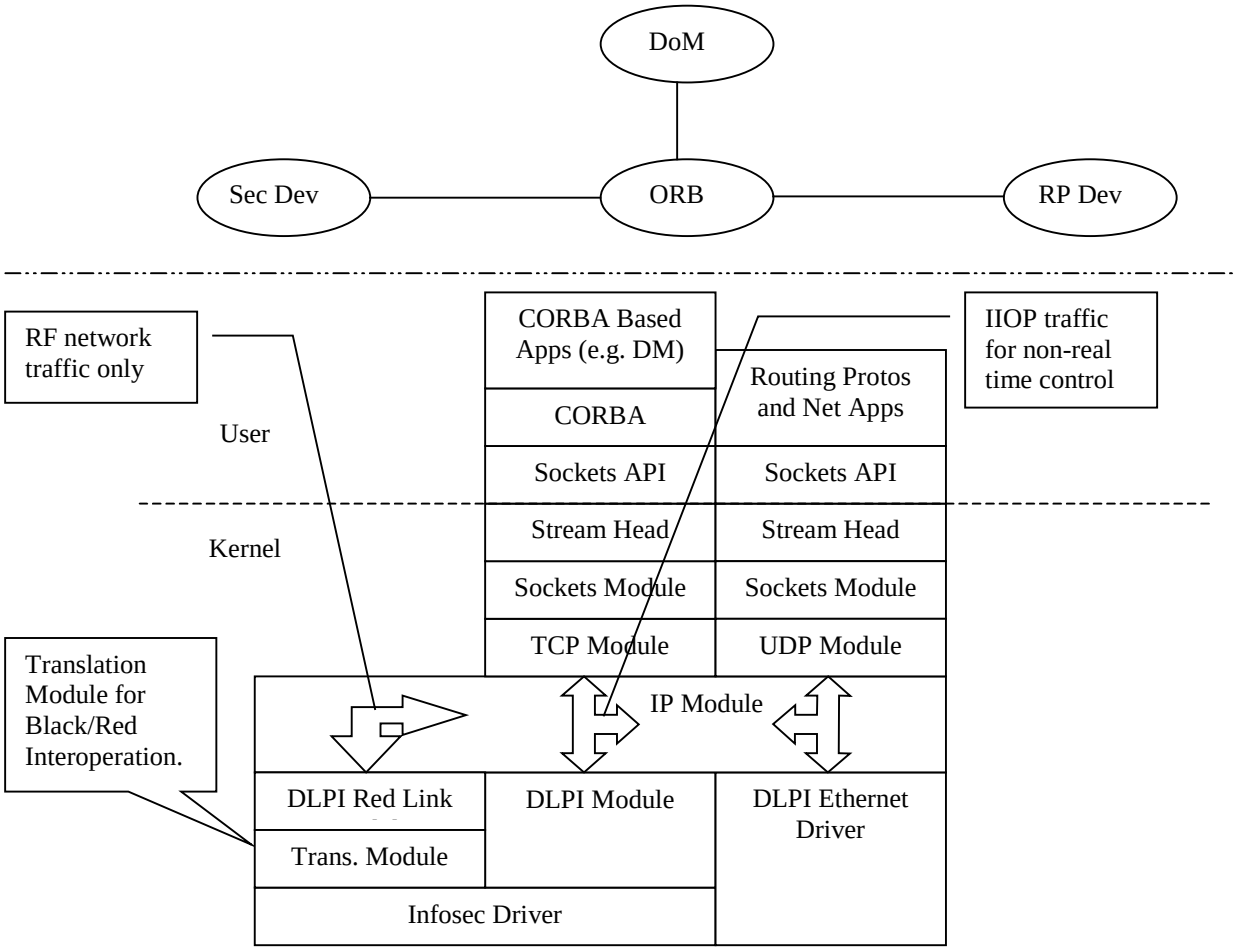


Figure 9 Red Side Stack with Translation Module

3.2.4.6.7 Cost

The following is some information from the Solaris Operating System to provide a frame of reference for the size of protocol modules in general and STREAMS modules in particular. The following table was generated by applying a byte to line of code byte estimate of 10-20.

Protocol Module	***Size in bytes***	***Approx. SLOC***
IP	*122227*	*6k-12 k*
TCP	*58044*	*2.9k-5.8k*
UDP	*8676*	*0.4k - 0.8k*
ICMP	*9488*	*0.45k – 0.9k*
ARP	*20151*	*1k – 2k*
Sockets Module	*35524*	*1.75k – 3.5k*
Hme network interface	*48058*	*2.4k-4.8 k*

3.2.4.6.7.1 Implementation Cost

Reduces level of custom development because COTS stack implementations are available for STREAMS. The implementation cost in most cases would be limited to link and modem. Implementation of transport layers will happen less often. Commercial vendors offer the source code for STREAMS modules including link layer skeletons allow for reuse of basic functions in developed modules.

An investigation of one vendor's license fees for STREAMS yielded the following information:

Source Code License	**Per Unit Cost**	**Per 10,000 Units Cost**
$75,000	$400	$25

3.2.4.6.7.2 Maintenance Cost

- *See Porting Analysis:Migration of protocol standards.*

3.2.4.6.7.3 Porting Cost

- *See Porting Analysis: .Environment Change*

3.2.4.6.8 Commercial Acceptance

3.2.4.6.8.1 Current Commercial Acceptance

3.2.4.6.8.1.1 STREAMS Standards

The OpenGroup publishes the following STREAMS based standards:

DLPI, Network Provider Interface (NPI) and Transport Provider Interface (TPI)

3.2.4.6.8.1.2 STREAMS Vendors

3.2.4.6.8.1.2.1 Mentat

Mentat specializes in providing networking products. Their STREAMS based products include:

- *A TCP/IP stack*
- *A portable STREAMS environment*
- *eXpress Transport Protocol (XTP)*
- *SkyX Gateway which provides high-speed Internet access over satellites.*

Their customer base includes Apple, Compaq, HP, IBM, Lucent, Motorola, Novell, Sony, Sun, Wind River and Xerox.

3.2.4.6.8.1.2.2 Spider

Spider specializes in providing STREAMS based products. These include:

- *A STREAMS environment that can run on Unix, RTOSes or with no OS at all.*
- *A Distributed STREAMS Function (DSF)*
- *A Full TCP/IP stack*

- *An OSI stack (layers 1-4)*
- *A Q3 Telecom Stack*
- *X.25, PPP, ISDN, Frame Relay, and CMIP and SS7 products*

Their customer base includes Motorola, Lucent, Alcatel, Nortel, Ericsson, IBM, Hewlett Packard, Sun Microsystems, ICL, NEC, Compaq, Artesyn, PTI, SBE, BAe Sema, Raytheon, Tandem and Stratus.

3.2.4.6.8.1.2.3 StreamSoft

StreamSoft specializes in providing LAN-WAN connectivity products. Their STREAMS products include:

- *A Portable STREAMS environment.*
- *A Full TCP/IP stack*
- *X.25, PPP and Frame Relay products*

Their customer base includes QWEST Communications, Nextel communications, Cable and Wireless, Ericsson, Nokia, Marcatel, Tellabs and Servei de Telecomunicacions d'Andorra.

3.2.4.6.8.1.2.4 UCONx

UCONx specializes in providing serial connectivity products. Their STREAMS products include:

- *A STREAMS-based real-time operating environment*
- *Various serial protocols*
- *Frame Relay*

3.2.4.6.8.1.2.5 Gcom

Gcom provides connectivity products. STREAMS bases products include:

Frame Relay, X.25, SDLC/HDLC, LAPB, LAPD, LLC

3.2.4.6.8.1.2.6 Adax

Adax offers three basic product lines. The advanced Protocol Controllers (APC) support data link layer and protocol functions. The Advanced Network Controllers (ANC) operate at the physical layer. The advanced Protocol Software (APS) products are implemented at both the network and data link layers.

Internally, standard AT&T UNIX Streams-based intermodular communications support interoperability among Adax products. This means that, where it is logical, products at any layer can be combined with products at the other layers to build a fully functional communications solution.

The Advanced Protocol Software products include the drivers for each APC card and the software modules that support Internet connectivity and general WAN access via ATM, Frame Relay, PPP, X.25, SSCF/SSCOP, SS7 MTP-2, LAPB/D, or ISDN (Q.931) on Windows NT or

UNIX. All drivers implement a STREAMS-based DLPI programming interface, while the software modules are all STREAMS-based, pushable modules that can be seamlessly interconnected for flexible and efficient protocol stack construction.

Adax's customer base includes Lucent, Ericcson, Direct Connect, Novell, Sun, IBM, Trillium, HP and Hughes Software Systems.

3.2.4.6.8.1.2.7 Litton

Uses STREAMS as the host interface for TCIM modules.

3.2.4.6.8.1.3 Operating Systems with Native STREAMS

Solaris, HP-UX, Apple Mac OS, Concurrent PowerMAX OS, Sony NEWS and any ATT SVR4 compatible OS

3.2.4.6.8.1.4 Operating Systems Third Party STREAMS

VxWorks, Lynx, Linux, Windows NT, pSOS, QNX, VRTX, RTEMS

3.2.4.6.8.2 Future Commercial Acceptance

As technology migrates it is likely that some other standard will gain acceptance. Currently for commercial high bandwidth networking there are implementations of IP routing in hardware.

3.2.4.6.9 Security

3.2.4.6.9.1 Can support Local/Global Address Space

STREAMS supports multi-homing and parallel stacks. Figure 10 shows a local/global stack implementation using STREAMS.

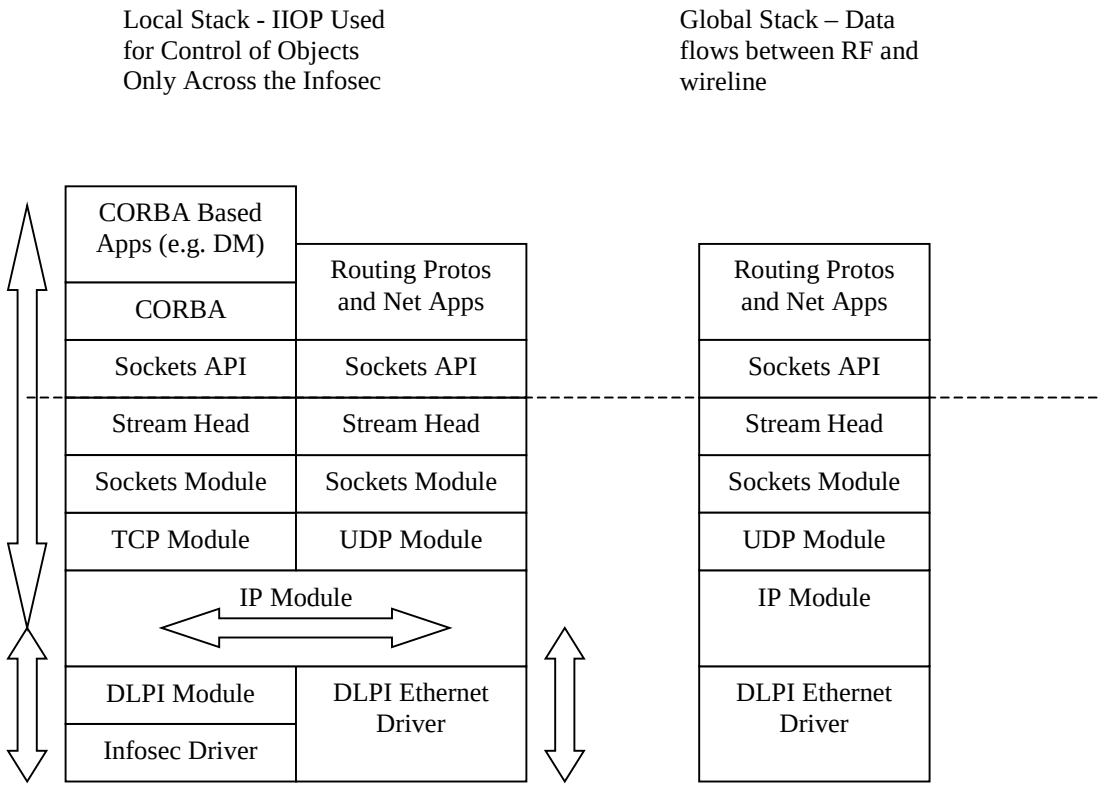


Figure 10 Two Stacks for Separate Local/Global Addressing

3.2.4.6.9.2 Supports Infosec requirements

This is a binary. Must identify the Infosec requirements.

3.2.4.6.9.3 How does transfer mechanism handle control/status, header and plain text bypass?

3.2.4.6.9.3.1 Non-Realtime Control/Status

IIOP is passed through the INFOSEC clear.

3.2.4.6.9.3.2 Header Bypass (Realtime Control/Status)

STREAMS identifies the following message blocks:

- *M_PROTO – This identifies the information as protocol control/status*
- *M_PCPROTO – This identifies the information as high priority protocol control/status*
- *M_DATA – This identified the information as data.*

Essentially the M_PROTO and M_PCPROTO blocks would be passed clear and the M_DATA blocks would be encrypted/decrypted.

The logical device for this interface is different than for the Non-real-time control interface.

3.2.4.6.9.3.3 Plain Text Bypass

The plain text bypass for the waveform is implemented in the INFOSEC.

3.2.4.6.10 Ability to Adapt to Commercial Trends

Identify how the given transfer mechanism could support moving to other implementations.

Unknown.